Titre: Title:	Planning and Management of Cloud Computing Networks
Auteur: Author:	Federico Larumbe
Date:	2013
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Larumbe, F. (2013). Planning and Management of Cloud Computing Networks [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1283/

UNIVERSITÉ DE MONTRÉAL

PLANNING AND MANAGEMENT OF CLOUD COMPUTING NETWORKS

FEDERICO LARUMBE
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE ÉLECTRIQUE)
SEPTEMBRE 2013

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

PLANNING AND MANAGEMENT OF CLOUD COMPUTING NETWORKS

présentée par: LARUMBE Federico
en vue de l'obtention du diplôme de: Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de:

M. GIRARD André, Ph.D., président
Mme SANSÒ Brunilde, Ph.D., membre et directrice de recherche
M. EL HALLAOUI Issmail, Ph.D., membre
M. GRÉGOIRE Jean-Charles, Ph.D., membre

To my family

ACKNOWLEDGMENTS

This thesis has been possible thanks to the strong support of my research advisor Brunilde Sansò that shared her knowledge, motivation, and resources as well as the creation of a lab with excellent people. I appreciate very much the availability and valuable comments of the jury André Girard, Issmail El Hallaoui, and Jean-Charles Grégoire. I am also very thankful for the help of my colleagues and friends Marnie Vodounou, Lê Nguyen, and Hadhami Dbira. This thesis has also been possible thanks to the Group for Research in Decision Analysis (GERAD) and Polytechnique Montréal that provided an environment with very bright people in optimization and telecommunication.

ABSTRACT

The evolution of the Internet has a great impact on a big part of the population. People use it to communicate, query information, receive news, work, and as entertainment. Its extraordinary usefulness as a communication media made the number of applications and technological resources explode. However, that network expansion comes at the cost of an important power consumption. If the power consumption of telecommunication networks and data centers is considered as the power consumption of a country, it would rank at the 5^{th} place in the world. Furthermore, the number of servers in the world is expected to grow by a factor of 10 between 2013 and 2020. This context motivates us to study techniques and methods to allocate cloud computing resources in an optimal way with respect to cost, quality of service (QoS), power consumption, and environmental impact. The results we obtained from our test cases show that besides minimizing capital expenditures (CAPEX) and operational expenditures (OPEX), the response time can be reduced up to 6 times, power consumption by 30%, and CO_2 emissions by a factor of 60.

Cloud computing provides dynamic access to IT resources as a service. In this paradigm, programs are executed in servers connected to the Internet that users access from their computers and mobile devices. The first advantage of this architecture is to reduce the time of application deployment and interoperability, because a new user only needs a web browser and does not need to install software on local computers with specific operating systems. Second, applications and information are available from everywhere and with any device with an Internet access. Also, servers and IT resources can be dynamically allocated depending on the number of users and workload, a feature called elasticity.

This thesis studies the resource management of cloud computing networks and is divided in three main stages. We start by analyzing the planning of cloud computing networks to get a comprehensive vision. The first question to be solved is what are the optimal data center locations. We found that the location of each data center has a big impact on cost, QoS, power consumption, and greenhouse gas emissions. An optimization problem with a multi-criteria objective function is proposed to decide jointly the optimal location of data centers and software components, link capacities, and information routing.

Once the network planning has been analyzed, the problem of dynamic resource provisioning in real time is addressed. In this context, virtualization is a key technique in cloud computing because each server can be shared by multiple Virtual Machines (VMs) and the total power consumption can be reduced. In the same line of location problems, we propose a Green Cloud Broker that optimizes VM placement across multiple data centers. In fact,

when multiple data centers are considered, response time can be reduced by placing VMs close to users, cost can be minimized, power consumption can be optimized by using energy-efficient data centers, and CO_2 emissions can be decreased by choosing data centers provided with renewable energy sources.

The third stage of the analysis is the short-term management of a cloud data center. In particular, a method is proposed to assign VMs to servers by considering communication traffic among VMs. Cloud data centers receive new applications over time and these applications need on-demand resource provisioning. Each application is composed of multiple types of VMs that interact among themselves. A program called scheduler must place each new VM in a server and that impacts the QoS and power consumption. Our method places VMs that communicate among themselves in servers that are close to each other in the network topology, thus reducing communication delay and increasing the throughput available among VMs. Furthermore, the power consumption of each type of server is considered and the most efficient ones are chosen to place the VMs. The number of VMs of each application can be dynamically changed to match the workload and servers not needed in a particular period can be suspended to save energy.

The methodology developed is based on Mixed Integer Programming (MIP) models to formalize the problems and use state of the art optimization solvers. Then, heuristics are developed to solve cases with more than 1,000 potential data center locations for the planning problem, 1,000 nodes for the cloud broker, and 128,000 servers for the VM placement problem. Solutions with very short optimality gaps, between 0% and 1.95%, are obtained, and execution time in the order of minutes for the planning problem and less than a second for real time cases. We consider that this thesis on resource provisioning of cloud computing networks includes important contributions on this research area, and innovative commercial applications based on the proposed methods have promising future.

RÉSUMÉ

L'évolution de l'internet a un effet important sur une grande partie de la population mondiale. On l'utilise pour communiquer, consulter de l'information, travailler et se divertir. Son utilité exceptionnelle a conduit à une explosion de la quantité d'applications et de ressources informatiques. Cependant, la croissance du réseau entraîne une importante consommation énergétique. Si la consommation énergétique des réseaux de télécommunications et des centres de données était celle d'un pays, il se classerait 5^e pays du monde. Pis, le nombre de serveurs dans le monde devrait être multiplié par 10 entre 2013 et 2020. Ce contexte nous a motivé à étudier des techniques et des méthodes pour affecter les ressources d'une façon optimale par rapport aux coûts, à la qualité de service, à la consommation énergétique et à l'impact écologique. Les résultats que nous avons obtenus minimisent les dépenses d'investissement (CAPEX) et les dépenses d'exploitation (OPEX), réduisent d'un facteur 6 le temps de réponse, diminuent la consommation énergétique de 30% et divisent les émissions de CO_2 par un facteur 60.

L'infonuagique permet l'accès dynamique aux ressources informatiques comme un service. Les programmes sont exécutés sur des serveurs connectés à l'internet, et les usagers peuvent les utiliser depuis leurs ordinateurs et dispositifs mobiles. Le premier avantage de cette architecture est de réduire le temps de mise en place des applications et l'interopérabilité. En effet, un nouvel utilisateur n'a besoin que d'un navigateur web. Il n'est forcé ni d'installer de programmes sur son ordinateur, ni de posséder un système d'exploitation spécifique. Le deuxième avantage est la disponibilités des applications et de l'information de façon continue. Celles-ci peuvent être utilisées à partir de n'importe quel endroit et de n'importe quel dispositif connecté à l'internet. De plus, les serveurs et les ressources informatiques peuvent être affectés aux applications de façon dynamique, selon la quantité d'utilisateurs et la charge de travail. C'est ce que l'on appelle l'élasticité des applications.

Cette thèse étudie l'allocation des ressources des réseaux infonuagiques et elle est divisée en trois étapes principales. Nous analysons en premier lieu la planification des réseaux infonuagiques pour avoir une vision complète. La première question à résoudre concerne la localisation optimale des centres de données. Nous montrons que leur position a un impact important sur les coûts, sur la qualité de service, sur la consommation énergétique et sur les émissions de gaz à effet de serre. Nous proposons un problème d'optimisation avec une fonction à plusieurs objectifs pour décider simultanément des positions des centres de données, de celles des applications, des capacités des liens et du routage de l'information.

La deuxième étape étudie le problème de l'allocation dynamique des ressources en temps

réel. Dans ce contexte, la virtualization est une technique très importante parce qu'elle permet d'utiliser un seul serveur comme support physique de plusieurs machines virtuelles (MVs). La consommation énergétique totale est alors réduite. Dans cette optique, on propose aussi un système qui optimise le positionnement des MVs sur plusieurs centres de données. Ainsi, lorsque l'on considère plusieurs centres de données, on constate que le temps de réponse est réduit quand les MVs sont placées près des utilisateurs, que le coût est réduit pour des fournisseurs meilleur marché, la consommation énergétique est moindre pour les centres de données ayant une bonne utilisation de l'énergie, et les émissions de CO_2 peuvent être réduites en choisissant des centre de données utilisant une énergie verte.

La troisième étape de l'analyse porte sur la gestion des ressources d'un centre de données. Une méthode est proposée pour affecter des MVs aux serveurs en tenant compte de la communication entre les MVs et les variations de la charge de travail. Un centre de données reçoit des applications les unes après les autres, et doit satisfaire leurs demandes en ressources. Chaque application est composée de plusieurs types de MVs qui interagissent entre elles. Un programme appelé ordonnanceur doit placer chaque nouvelle MV sur un serveur. Ceci a un impact sur la qualité de service et la consommation énergétique. La méthode proposée place les MVs devant communiquer entre eux près les unes des autres. Ainsi, le délai de communication est réduit et le débit disponible entre les MVs est augmenté. De plus, la consommation énergétique de chaque type de serveur est prise en compte et les serveurs les plus efficaces sont choisis pour héberger les VMs. La quantité de MVs de chaque application est modifiée de façon dynamique selon la charge de travail et les serveurs qui ne sont pas nécessaires à un moment donné sont mis en veille pour épargner de l'énergie.

La méthodologie développée se base sur des modèles de programmation en nombre entiers pour formaliser les problèmes et pour utiliser les solveurs à l'état de l'art. Des heuristiques sont développées pour résoudre des cas avec plus de 1,000 centres de données possibles pour le problème de planification, plus de 1,000 noeuds pour le problème d'affectation de MVs aux centres de données et 128,000 serveurs pour l'affectation de MVs aux serveurs. Les solutions trouvées ont des intervalles d'optimalité très petits, entre 0 et 1.95%. Le temps d'exécution est de l'ordre de quelques minutes pour le problème de planification et moins d'une seconde pour les cas en temps réel. Nous considérons que cette thèse sur l'allocation de ressources des réseaux infonuagiques apporte des contributions importantes à ce domaine de recherche, et des applications commerciales basées sur les méthodes proposées ont un avenir prometteur.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABREVIATIONS

API	Application Programming Interface
AS	Autonomous System
CAPEX	Capital expenditures
CDN	Content Delivery Network
CNPP	Cloud Network Planning Problem
CPU	Central Processing Unit
CS	Cloptimus Scheduler
DB	Database
DCLP	Data Center Location Problem
DVFS	Dynamic Voltage and Frequency Scaling
FF	First Fit
FIFO	First in First Out
GCBP	Green Cloud Broker Problem
HPC	High Performance Computing
ICT	Information and Communications Technology
ISP	Internet Service Provider
IXP	Internet Exchange Point
IaaS	Infrastructure as a Service
LAN	Local Area Network
LHC	Large Hadron Collider
MCDA	Multi Criteria Decision Analysis
MC	Memcache

MILP	Mixed Integer Linear Programming
MIP	Mixed Integer Programming
MSE	Mean Square Error
NASA	National Aeronautics and Space Administration
OPEX	Operational expenditures
OXC	Optical cross-connect
PUE	Power Usage Effectiveness
PaaS	Platform as a Service
rps	Requests per second
QAP	Quadratic Assignment Problem
QoE	Quality of experience
QoS	Quality of service
SLA	Service Level Agreement
SaaS	Software as a Service
ToR	Top of Rack
UI	User Interface
VDC	Virtual Data Center
VM	Virtual Machine
WS	Web Service

CHAPTER 1

INTRODUCTION

In 1963, researchers discussed the possibility of connecting some research institutes' computers to share resources, programs, and information. Licklider [5] called it the *Intergalactic Computer Network*, and in the same memo that describes the network, the author asked if a program to plot functions should be downloaded from a remote computer and executed locally, or if the data should be sent to the remote computer that stores the program, execute it there, and then download the results. This shows that location of data and programs has been an architectural issue for a long time. Some years later, in 1969, the two first nodes of the ARPANET, precursor of the Internet, were connected. Though the technology was at its beginnings, its creators already envisioned that computing could be accessed as an utility from homes and offices in the same way as electricity and telephone [6]. Over the years, the Internet has expanded and computers have increased their capacity exponentially. At the end of the 1980s, scheduling algorithms were developed to execute tasks on multiple processors, a concept known as *High Performance Computing (HPC)*. By the late 1990s, grid computing became popular when general middlewares executed HPC applications on multiple independent clusters [7]. At the same time, the world wide web rapidly arose and massive web sites became available. Internet data centers grew and multiple commercial models appeared to provide computing resources.

The concept of providing computing as an utility is currently known as *cloud computing* [8]. Cloud services can be classified in three service models depending on what are the actions of clients and providers. In *Infrastructure as a Service (IaaS)*, a client pays for accessing raw IT resources managed by a provider: processing power, network bandwidth, storage. The client is responsible for maintaining the software, and the infrastructure provider manages the hardware and the computing facility. In *Platform as a Service (PaaS)*, a provider installs commonly used software such as operating systems, programming environments, databases, and load balancers. Then, each client develops applications in a programming environment such as Python, Java, and .NET in the given platform. The third cloud computing model is called *Software as a Service (SaaS)*, and it is what we use everyday. The client uses applications given by the provider such as e-mail, social networks, spreadsheets, word processors, and photo editors.

Providers store data and execute applications in computers hosted in facilities called *data centers*. Users access applications from their computers and mobile devices. The Internet

provides access between client devices and servers composed of cellular antennas, wireless routers, modems, routers, optical links, and optical cross connects.

1.1 Research objectives

In the present thesis, we propose planning and management methods to improve the following aspects of cloud computing networks:

Quality of Service (QoS)

As people use more cloud applications, high availability and short response time become increasingly important. The *response time* is composed of the server execution time and the time to send information between servers and client devices. The first one depends on the server capacity and the complexity of the program executed. The *communication delay* is associated to the transmission delay, queuing delay, and propagation delay. The *transmission delay* is inversely proportional to the network interface capacity. The *queuing delay* is the time that information packets wait while other packets are being served in each router between the client and the server. The *propagation delay* is the time that electrical and optical signals take to travel each link.

Cost

In the planning and management of cloud computing networks, cost is a fundamental metric [9]. Providers are interested in paying the least possible while achieving the expected QoS. The cost is composed of capital expenditures (CAPEX) and operational expenditures (OPEX). *CAPEX* are costs that create future benefits, including servers, routers, optical links, and data centers. *OPEX* are the ongoing costs for running the system: electricity, the water used for cooling, IT staff, security guards, administrative staff, and external providers. Costs can be minimized with a planning that reduces the amount of resources needed, and an efficient resource management to reduce power consumption.

Power consumption

Each network element consumes power over time. If the power consumption of global telecommunication networks and data centers was considered as the power consumption of a country, it would rank as the 5^{th} country of the world, as can be seen in Figure 1.1. More precisely, a large data center can consume the same energy as 80,000 American households or 250,000 European ones [1]. Given that projections predict a multiplication of the number of servers

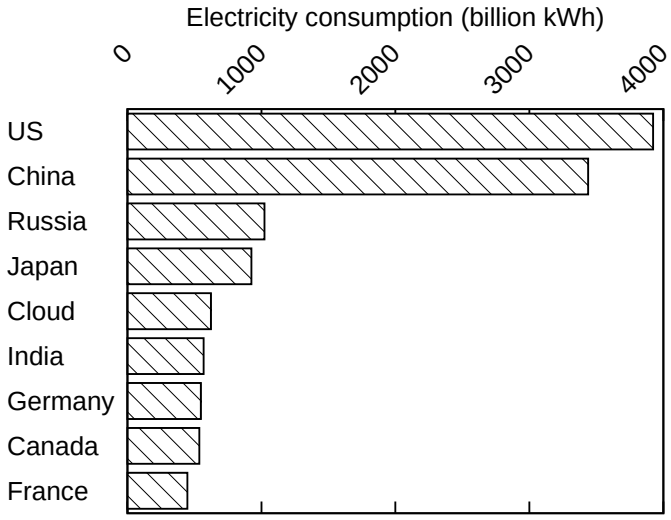

Figure 1.1 Energy consumption of countries and the cloud. Adapted from Cook [1].

by a factor of 10 between 2013 and 2020 [10], resources must be optimally allocated to applications in order to limit the growth of power consumption.

Greenhouse gas emissions

Besides controlling power consumption, reducing CO_2 emissions is also needed given their impact on global warming. In fact, the method of power generation has a big impact on the amount of CO_2 emitted: 10 g of CO2 per kWh are produced by wind and hydroelectric power, 38 g / kWh by geothermic energy, 66 g / kWh by nuclear power, 778 g / kWh by diesel, and 960 g / KWh by coal [11]. Therefore, the type of energy used greatly impacts the greenhouse gas emissions, and these emissions increase the global warming.

1.2 Problems studied

This thesis optimizes the objectives described above in multiple stages: 1) planning the network of data centers, 2) coordinated management of multiple data centers, 3) internal management of each data center. As we will see, the proposed models have a common structure but the decisions in each of them are taken at different times and have different duration.

Cloud Network Planning Problem (CNPP)

The method first developed for this thesis optimizes the planning of an infrastructure provider network. More precisely, we study how data center location impacts on cost, quality of

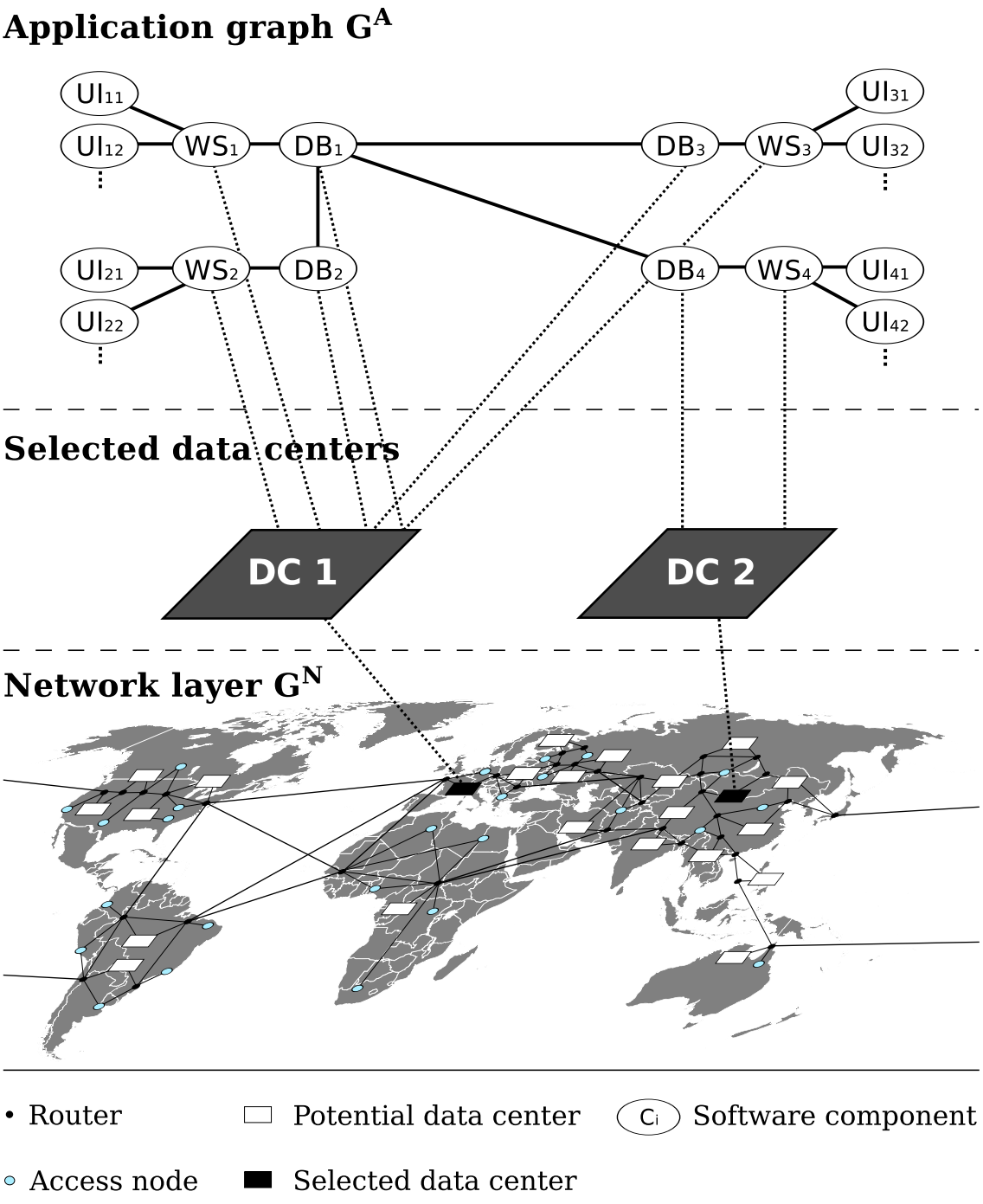

Figure 1.2 Data center location and software component placement.

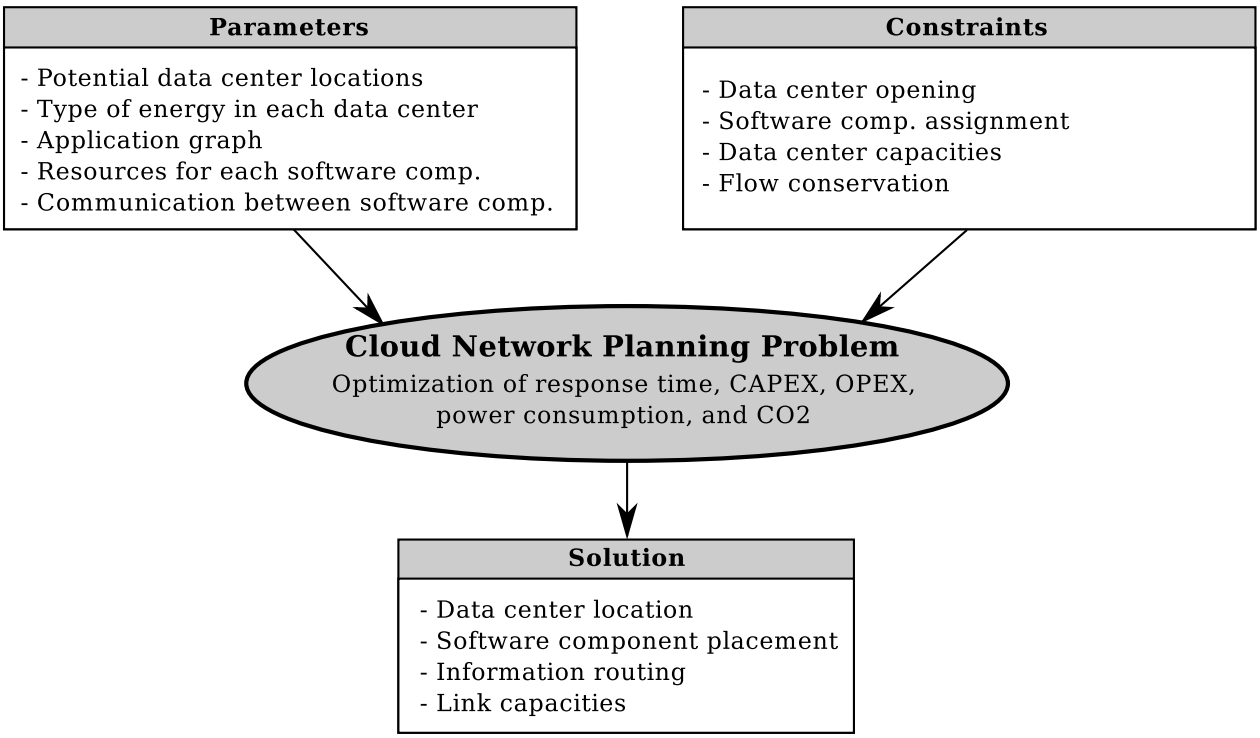

Figure 1.3 CNPP parameters, constraints, and solution.

service (QoS), power consumption, and CO_2 emissions. In fact, cost is reduced when data centers with low energy and land prices are chosen, the QoS is improved when data centers are placed close to users, power consumption is reduced with data centers in cold climate, and CO_2 emissions are dramatically reduced in regions where green energy is available. A Mixed Integer Linear Programming (MILP) model is presented with an objective function that combines multiple criteria with priority coefficients. Planners can change these coefficients to analyze multiple solutions making trade-offs that depend on their priorities.

We state the problem as a mapping between two graphs: an application layer G^A and a network topology G^N. Each node in G^A is a software component, and each arc (i, j) represents the fact that software component i sends information to software component j. The network layer is composed of access nodes, backbone routers, potential data center locations, and the links that connect these nodes. The optimization model must decide for each software component in G^A, which node in G^N it is assigned to. Then, the nodes in G^N that will host software components must be open, e.g., if a solution places software components in a data center, then this data center is selected. Each software component may require to be executed in multiple servers (e.g., a thousand) and there is a capacity constraint in each data center. Also, each communication demand in G^A represents a flow of information packets that must be routed through G^N.

Figure 1.2 shows an example of a global application provider with users in four different regions: 1) North America, 2) Latin America, 3) Europe and Africa, and 4) Asia and the Pacific. The network layer is composed of potential data centers, backbone routers, and access nodes that aggregate users. The application is composed of a user interface that is executed on the client computers, a web service, and a database. The application is further split by region, as shown at the top of the figure. There is one web service responding to the users in each region, a database replica for each region, and a user interface for each access node. The arcs of the application graph are communication demands between the software components: traffic between the user interfaces and web services, between the web services and databases, and between the databases and master database. The problem consists of selecting a subset of the potential data centers and deciding which data center will host each web service and database. Furthermore, each web service is executed on multiple servers. Thus, placing a software component in a data center means that we must also allocate servers for that purpose without exceeding the data center capacity. The solution for this example was to select two of the potential data centers: one in Europe and one in Asia. The users from North America, Latin America, and Europe are served from the data center located in Europe, and the users in Asia obtain cloud services from the data center located in that region.

The comprehensive planning problem summarized in Figure 1.3 simultaneously answers the following questions:

- Among a set of potential data center locations, which ones should be used to open new data centers?
- Which data center should host each of the software components of the cloud applications?
- How many servers will be hosted at each data center?
- How will the information be routed through the network?
- What are the link capacities required to carry that information?

The MILP model was first solved with AMPL/Cplex for cases with up to 12 access nodes and 24 potential data center locations. An exhaustive research was done to define all the parameters based on realistic values and public data [9]. The relation between cost, response time, power consumption, and CO_2 emissions was studied for that kind of networks. However, the case of 24 data center locations took 58 minutes to solve, and we were interested in solving cases with 1,000 data center locations because the largest global networks have that kind of size [12]. Thus, we developed a tabu search heuristic [13] to handle large cases. Comparing the solutions of the small cases to the optimal solutions gave as a result optimality gaps between 0 and 1.95%, and most of them were below 0.1%. The case of 24 potential data centers was solved by the heuristic in 868 milliseconds, while Cplex took 58 minutes. The case of 1,000 potential data centers was solved in 10 minutes and was unsolvable in Cplex.

As we will see in Chapter 2, few authors specifically addressed the problem of data center location. Our work first differentiates in the flexible representation of each application as a graph of software components. That representation precisely defines the communication pattern of any kind of applications, that is very valuable to plan distributed applications. Our model also uses a detailed calculation of cost, communication delay, power consumption, and CO_2 emissions, and it is also flexible to allow planners to define priorities. Furthermore, even if it is difficult to compare our technique with other work, because they solve different models, our heuristic took 21 seconds to solve an instance with 500 data centers and the work most closely related [14] solved a case with the same number of data centers in 30 minutes in a similar environment.

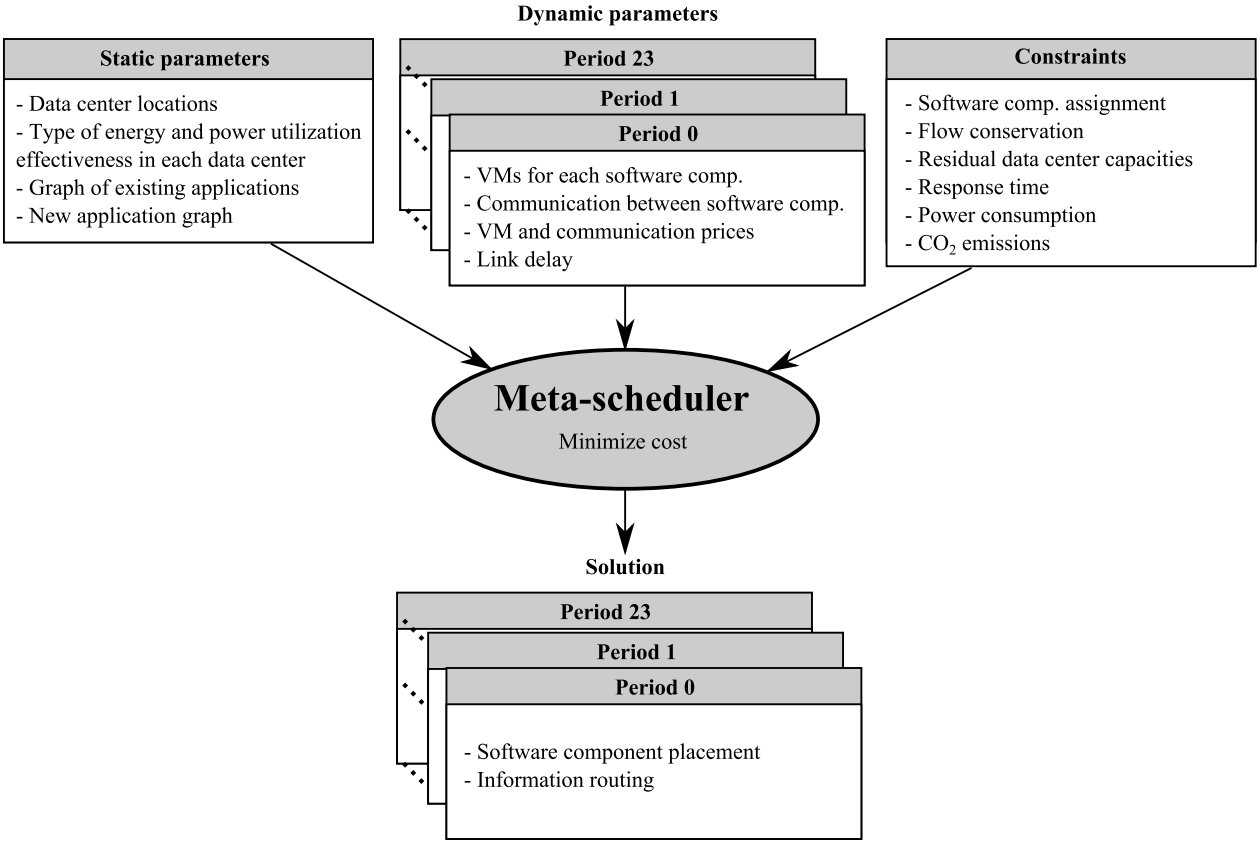

Figure 1.4 Meta-scheduler parameters, constraints, and solution.

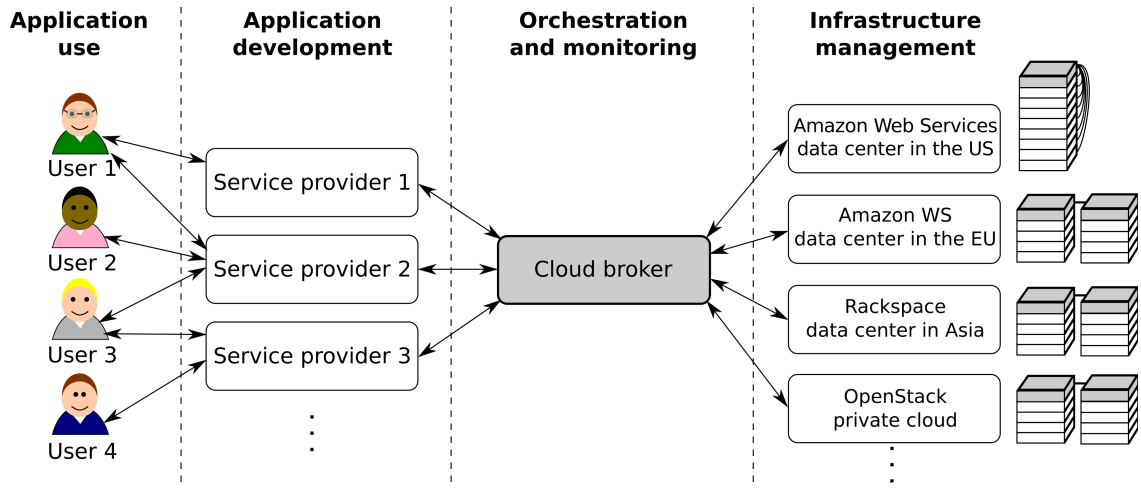

Figure 1.5 Actors and components interacting in the life-cycle of cloud applications.

Meta-scheduling: resource allocation across multiple data centers

Once the network is defined, the every day management of the cloud computing system has also an impact on cost, QoS, power consumption, and CO_2 emissions. In the same line of location problems, we analyzed the online placement of software components across multiple data centers, a process called *meta-scheduling*. In this case, the system has a set of data centers already open, and applications arrive consecutively. Furthermore, the application workload presents daily fluctuations with periods of high and low utilization. The dynamic provisioning of resources, or dynamic scaling, allocates resources needed by each application at each time instead of reserving a large amount of resources for peak periods.

That dynamism can be well managed through virtualization technique [15], where each physical server executes one or multiple Virtual Machines (VMs). In this case, instead of using each server for a single purpose, it can host different VMs over time: A web service VM during the day to serve users' HTTP requests and a HPC application overnight to process big amounts of data and generate statistics. In fact, each software component is executed on a different number of VMs in each period of the day, and all the VMs use the same shared pool of servers in a data center.

The problem that is solved at this stage takes place at the time of deploying a new application. A meta-scheduler must choose a data center to host the VMs of each software component taking into account dynamic workload variations. That impacts the cost because different data centers have different VM prices. It also impacts the power consumption because the servers used for the VMs will consume energy, and each data center has a *Power Usage Effectiveness (PUE)* that accounts for the overhead of cooling and power distribution. The CO_2 emissions change depending on the type of energy that provides the data center.

As summarized in Figure 1.4, this optimization problem minimizes costs by answering the following question:

- Which data center will host each software component's VMs in each period of the day, given the expected workload, expected performance, and current network state?

The problem was stated from the point of view of a *cloud broker*, that is, an intermediate entity between the service providers that need to execute applications and the infrastructure providers that offer computing capacity, as can be seen in Figure 1.5. A cloud broker is a program that manages resources of multiple data centers belonging to one or more providers in a unique platform. Commercial cloud brokers exist—e.g., RightScale, Enstratius, Gravitant, and Scalr—, but they lack a meta-scheduler to help users optimally place software components. There is some work that addresses the meta-scheduler problem [16, 17, 18, 19].

Our proposal adds several contributions with respect to these approaches, including special considerations for communication traffic between users and VMs which is important for distributed cloud applications, multi-period dynamic scaling, and CO_2 emission reductions.

The solution approach is an online and multi-period heuristic based on a tabu search algorithm. The cases solved had more than 1,000 nodes, 650 applications, and 6,500 VMs. Results show that the application response time can be reduced up to 6 times as compared with using the cheapest data center, 30% of power consumption can be saved, and CO_2 emissions can be reduced by 60 times. The execution time to determine the optimal software component placement of an application took between a few milliseconds to less than a second, making the program suitable to be embedded in an interactive cloud broker platform.

VM placement within a data center

A third stage in the process of resource provisioning of cloud computing networks is the management of each particular data center. In this context, the problem to solve is which server should host each VM. Figure 1.6 summarizes the features of the proposed Mixed Integer Programming (MIP) model. The objective is to minimize response time and power consumption subject to capacities of server resources. Similarly to the previous two stages, we emphasize that the VMs of an application can communicate among themselves. VMs that communicate among themselves can be placed in nearby servers to reduce delay and increase the bandwidth globally available among VMs.

In this problem, the dynamic aspect is very important. Cloud providers receive new applications over time, and a program called *scheduler* must allocate resources to each application. That is, when a new application arrives, each VM must be placed in one of the servers. Each server can host multiple VMs, and the servers that are not hosting VMs can be put in sleep mode to save power. Furthermore, the workload of an application can change over the day, with periods of high and low activity, so that the number of VMs in an application can be changed to match the dynamic workload.

Many authors have studied the problem of VM placement within a data center, but only a few of them took special attention to the communication among VMs [20, 21, 22]. Our contribution is a method that considers both communication among VMs and a dynamic workload. We take advantage of the hierarchic topology of data centers to propose a hierarchic tabu search heuristic that can scale to more than 100,000 servers and VMs.

The test cases analyzed show encouraging results: the transmission delay is reduced by 70% and link utilization decreases up to 33% compared to methods that do not consider traffic; the power consumption is reduced by 4.9% representing $1.6 millions savings per year in a data center with 128,000 servers. This means that our approach increases the quality

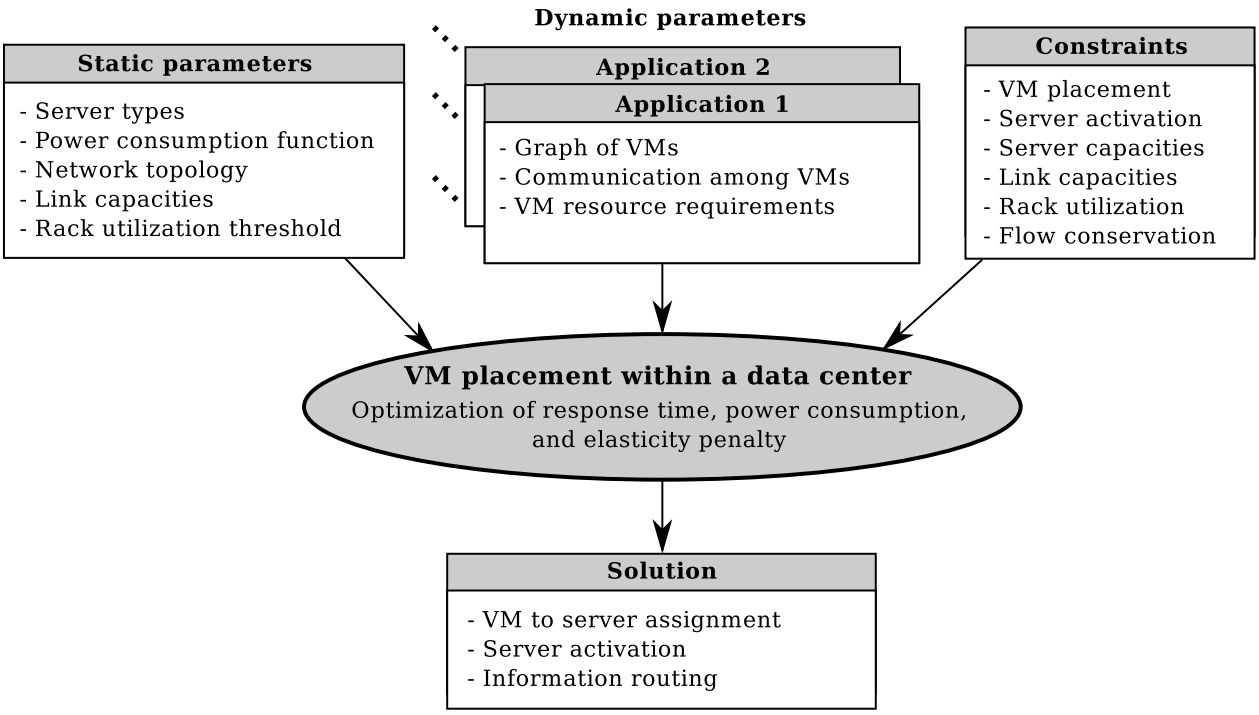

Figure 1.6 VM placement parameters, constraints, and solution.

of service by placing VMs that communicate among themselves in nearby servers, reducing delay and the utilization of links among switches.

Common structure

The three problems in planning and resource management have a common mathematical structure that is customized for each stage. Table 1.1 summarizes the main features of each problem that result in a mapping between an application graph G^A and a network G^N. That is, for each node in G^A, a node in G^N is chosen, and for each arc in G^A, network paths are used to carry the information. This is shown in the first two constraints of each problem in Table 1.1.

In the first stage, each node in G^A is a software component, and the nodes in G^N are potential data centers, access nodes, and routers. The potential data centers in G^N that host software components in a particular solution are the locations where new data centers will be opened. The communication delay is minimized and optimal links capacities are assigned, important aspects when planning a network. That is the reason why we include routing and link capacity assignment decisions. Given that when a location is chosen, a data center

Table 1.1 Three stage approach of cloud network planning and management.

Stage	Objective	Decisions	Constraints	Dynamics
1. Cloud network planning	- Cost - Response time - Power - CO_2 emissions	- Data center location - Software comp. placement - Routing - Link capacities	- 1 DC per SC - Flow conserv. - Data center capacities	- Static - Offline
2. Meta-scheduling	- Cost	- Software comp. placement - Routing	- 1 DC per SC - Flow conserv. - Data center capacities - Response time - Power - CO_2 emissions	- Dynamic workload - Dynamic prices - Online
3. VM placement within a data center	- Power - QoS	- VM placement - Server on/off switching - Routing	- 1 server per VM - Flow conserv. - Server capacities - Link capacities - Rack utilization	- Dynamic workload - Online

remains open in that location for several years, the planning problem is offline with static expected demands.

The second stage is to place applications across multiple data centers, that is, to choose the optimal data center for each software component. The graphs G^A and G^N are similar to the first stage with the difference that each software component requires a number of VMs, and we know which data centers are open. The decision of software component placement can change more often because deploying a software component is much less expensive that opening a data center. Furthermore, the workload of each software component changes over the day. That is the reason why we propose an online and dynamic optimization engine. To schedule a new application, residual data center capacities must be considered and the optimization is solved for multiple periods. In this case, the response time, power consumption, and CO_2 emissions were expressed as constraints to analyze how cost increases when the constraints are tighter.

Finally in stage three, the elements to place are the VMs that compose each application within a particular data center. In this case, each VM is placed in a server to minimize power consumption and improves the QoS. The workload changes over the day and the deployment of a new application must be done taking into account existing applications that are already being executed.

1.3 Contributions

The following is the list of contributions of this thesis:

1. A mixed integer programming framework for the location of cloud elements through a mapping between an application and a network graph.
2. Definition of the CNPP for planning cloud networks taking into account the cost, QoS, power consumption, and CO_2 emissions.
3. A tabu search heuristic for the CNPP that achieves near optimal solutions in short execution time.
4. Definition of the Green Cloud Broker Problem (GCBP) to produce the meta-scheduling of distributed applications across multiple data centers.
5. An online and multi-period heuristic for the GCBP that optimizes application deployments in less than a second.
6. Definition of a VM placement problem that handles traffic among VMs and elasticity of applications.
7. A hierarchic tabu search heuristic to schedule new applications and resize existing ones, handling cases of more than 100,000 servers and VMs in real time.

1.4 Document structure

The remainder of this document is structured as follows. Chapter 2 presents a literature review on cloud network planning and management. The articles closest to ours are analyzed as well as related problems of dynamic provisioning. Chapter 3 presents the paper Larumbe and Sansò [23] that defines the CNPP and the proposed tabu search heuristic. Chapter 4 presents the Green Cloud Broker for the dynamic placement of VMs across multiple providers as in Larumbe and Sansò [24]. Chapter 5 studies the dynamic placement of VMs in servers to optimize QoS and power consumption. Chapter 6 explains how the approaches can be integrated to achieve the objectives of energy consumption, pollution, quality of service, and cost minimization. Chapter 7 concludes the research on cloud network planning and dynamic provisioning and proposes future avenues of research and development.

CHAPTER 2

LITERATURE REVIEW: LOCATION AND PROVISIONING PROBLEMS IN CLOUD COMPUTING NETWORKS

This chapter presents a literature review on planning and management of cloud computing networks [1]. In this chapter, we discuss the problems mentioned in the introduction, analyze various methods to find optimal solutions, and present our contributions in each area. In Section 2.1, some definitions necessary to the understanding of the problems are presented. In that section, the general problem setting and relevant metrics are defined. Then, Section 2.2 discusses the Data Center Location Problem (DCLP) in the context of planning cloud computing networks. Section 2.3 describes dynamic scaling architectures and algorithms to determine the number of Virtual Machines (VMs) needed for each application in each period of the day. That section presents the concept of scheduling within a data center which is a prerequisite of meta-scheduling. Section 2.4 reviews articles about meta-scheduling to manage resources of multiple data centers in a coordinated fashion. Section 2.5 presents papers about the placement of VMs in servers within a data center. Finally, the chapter concludes with the current state of the location and provisioning problems in cloud computing.

2.1 Definitions and base concepts

This section is devoted to an overview on the common background of the problems that will be treated in this thesis. Such a background can be divided in three groups related to: 1) the applications and the computing environment, 2) the way energy is consumed and cost are evaluated and, finally, 3) users and performance issues.

2.1.1 Applications and computing background

Applications

Object-oriented programming is currently a very widespread programming paradigm. In this paradigm, a program is conceived as a set of objects that interact by sending messages. The objects can be hosted on a single computer or distributed across a computer network. When

[1] This chapter appears in *Communication infrastructures for cloud computing: design and applications* edited by Hussein Mouftah and Burak Kantarci. Copyright 2013, IGI Global, www.igi-global.com. Posted by permission of the publisher.

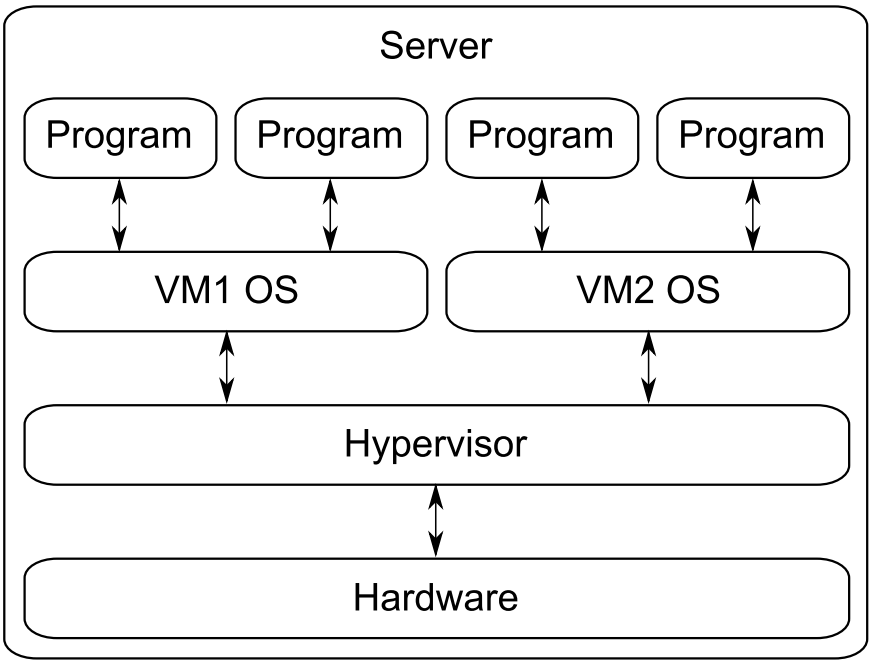

Figure 2.1 Server virtualization layout.

the objects are on the same computer, the messages are exchanged through the CPU and the RAM. Objects on different computers can communicate using protocols that convert the message into information packets, which are sent through links and routers connecting the two ends. In the client/server architecture, the program is split into a component that is executed on a client device and a component that is executed on a server. In practice, the client component is replicated, and multiple users can connect to the same server. Similarly, the server component can be replicated when a single server is not sufficient to efficiently process the user requests. The users can connect from their homes or offices using their phones, tablets or computers. The servers are hosted in large data centers to take advantage of economies of scale. Complex applications contain not only a client and a server but also a set of software components, each with its own specific purpose. For example, a popular social network can present the following set of software components: 1) a Javascript client that is executed in the user desktop browsers, 2) a client for smart phones, 3) a web server for answering HTTP requests, 4) a database that stores the user posts on hard disks, 5) a system to send and receive messages, and 6) file servers for picture storage. All these software components send messages to each other to accomplish the global purpose of the application.

Virtual machines

Between the software and hardware levels, there is an element known as the *Virtual Machine (VM)*. A VM is a program that simulates the behavior of a computer. Similar to a computer, it has access to processing units, RAM memory, hard disks, and devices. However, these elements are provided by a system known as the *hypervisor*, which is executed on the physical machine hosting the VM. As shown in Figure 2.1, any operating system can be installed on a VM, and the operating system is unaware that it is being executed on a virtual rather than a physical machine. The user programs interact with the operating system as usual. The hypervisor ensures that the VMs are isolated from each other and have the necessary resources. Using a virtualization framework offers many advantages to the cloud computing paradigm. An infrastructure provider can offer VMs to its customers and consolidate multiple VMs on a single server. Because many applications require fewer resources than an entire server, the server consolidation reduces the number of servers, cost, and energy consumption.

2.1.2 Cost and power consumption

CAPEX and OPEX

Each element of the cloud network has a purchasing cost, setup cost, and amortization period [9]. The server cost varies depending on the components: the CPU, capacity of the hard disks, RAM, and motherboard. In addition to servers, the routers, optical fiber, and optical repeaters contribute to cost, and the cost of the power distribution equipment, cooling equipment, data center building, and land must also be considered. To compute the total cost of all the elements, ranging from an individual server to the entire data center, the cost of each element is divided by its amortization period and then added to the sum. For instance, the amortization period of a server is between 3 and 4 years and that of a data center is between 12 and 15 years [9]. All the costs listed so far comprise the *capital expenditures (CAPEX)*. In addition to the CAPEX, there are the *operational expenditures (OPEX)*, including electricity, the water used for cooling, the IT staff to maintain the equipment, security guards, and administrative staff.

Power consumption

Each active network element consumes power. The power consumed by a server is the sum of the power consumed by its components. The Central Processing Unit (CPU) typically consumes the majority of the power, and depends on the degree of utilization. When the CPU is idle, the consumption is reduced. Figure 2.2 shows an example of the system power consumption as a function of its utilization [2]. The function is monotonically increasing, has

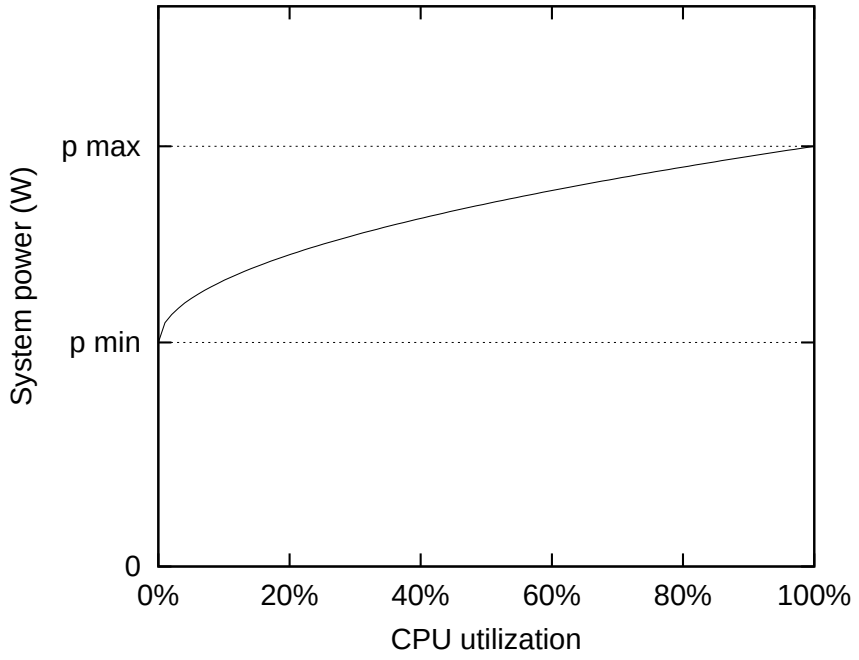

Figure 2.2 System power approximation. Adapted from Fan et al. [2].

a value of $p_{\min}$ when the utilization is zero and reaches $p_{\max}$ when the utilization is 100%. Depending on the server architecture, operating system, and hypervisor used, the consumption when the server is idle may be as high as 70% of the maximum power consumption. The consolidation of VMs in servers is therefore a promising strategy because suspending servers that are not highly utilized saves a large amount of power. The definite integral of the server power consumption over a given time interval is the total energy used during that period. The equipment used to distribute the power in the data centers has fixed limits on the total power consumption. For example, a row of 450 servers in a data center may have a maximum capacity of 150 KW. The active ports of the routers consume power, as do the optical cross-connects (OXCs) and optical repeaters. The power distribution and cooling equipment also consumes power, and the consumption rate determines the efficiency of the data center. The Power Usage Effectiveness (PUE) is defined as the ratio of the total power consumed by a data center to the power consumed by the IT equipment. For instance, a PUE of 1.15 means that the base equipment for cooling and power distribution requires 15% of the IT equipment requirement. The most efficient data centers may reach a PUE of 1.07 using mechanisms that exploit the surrounding climate for cooling. OpenCompute is a consortium of companies that promote the open design of efficient IT equipment and data centers by publishing the technologies used to achieve high data center efficiency [25].

In addition to the quantity of power consumed, one needs to pay attention to the power

generation mechanism: whether it is renewable and how much environmental pollution is introduced in the production process. Wind and hydroelectric power produce approximately 10 g of CO_2 per KWh. Geothermic energy production introduces 38 g of CO_2 per KWh, diesel produces 778 g / KWh, and coal produces 960 g / KWh [11]. The type of energy used by a data center therefore greatly impacts the pollution level.

2.1.3 Performance parameters

Response time and throughput

As we have seen, the software components communicate by exchanging messages among themselves. When a software component receives a message, we call it a request. The software component processes the request, potentially sending messages to other components, and answers with a message called a response. The time elapsed between the reception of the request and the response is called the *response time*. The inverse of that value is the *throughput*, which is the number of requests per second that a software component can process.

Communication delay

The messages exchanged between the software components are typically sent through a network protocol such as TCP and UDP over IP, which generates information packets that traverse the network. These packets pass through paths of routers and links between the source and destination hosts. Each router is connected to its links through network interfaces. When a router receives a packet, its header is analyzed, and then the packet is placed in the interface corresponding to the output link. The time of that operation is called the *processing delay*. If the output interface also had other packets to send, then the packet is placed in an output queue. The period between the reception of the packet in the output queue and the time when the packet is sent is called the *queuing delay*, and the time required to place each bit of the packet in the transmission channel is called the *transmission delay*. The time from the placement of the packet in the transmission channel to the moment when it is completely received on the other end is the *propagation delay*. That delay depends on the physical properties of the transmitting medium, e.g., the speed of light in optical fiber is approximately 200,000 km/s. The sum of all of these delays yields the delay of a segment router-link, and the sum of the delays of the segments in a network path is called the communication delay.

Availability

Hardware failures, software maintenance, and upgrades can prevent a service from responding to requests during specified periods. The *availability* is the proportion of time that a service is functional. For instance, we can say that the availability of an e-mail service was 99.99% in the last week. The same concept is applied to hardware elements such as servers, routers, and links. It can also be applied to an entire data center, e.g., the average availability of a data center may be 99.99% [9]. When applications are executed on multiple servers at the same time, the service availability increases because the failure of a single server has a lower impact. The availability can also be improved by avoiding single failure points in the network and using backup paths when a link or router fails.

Service Level Agreement (SLA)

The metrics discussed above impact the quality of service (QoS) that an operator provides to its customers. Customers and providers use *SLAs* to keep the metrics above (or below) a specified threshold. When the metric is outside their range, the provider must compensate the customer. A mechanism to monitor the metrics must also be defined, and tools must be developed to query and visualize the metrics in real time and over the history of the system. The models in this chapter typically aim to minimize the utilization of resources and satisfy the SLAs.

Workload forecasting

The models to plan and manage cloud computing networks require the workload size of each application as an input in order to determine the type and number of servers and VMs needed. The first issue that planners need to solve is to know the characteristics of the workload. Typically, there is a small number of applications that generate most of the traffic. Identifying these applications and their software components is needed. Then, for each application component, the workload can be split into different types of transactions depending on the objects exchanged, e.g., text, images, and videos.

Furthermore, the workload can change over time. The number of application requests per second over time can be displayed in a chart to analyze the workload. Some patterns that could arise are: increasing workload, decreasing, stationary, and/or with seasonal, weekly or daily fluctuations. For instance, a workload could increase over the year because new users are registered, and at the same time could vary over the day with peaks in specific hours.

There are two complementary classes of methods to forecast the workload of applications: qualitative and quantitative [26]. *Delphi* [27, 28] is a *qualitative method* where the expecta-

tions of actors in the organization and users are taken into account. In this case, business level metrics such as the number of orders per month are used to allow all the actors to participate in the forecast process. Then, the agreed forecast in terms of business metrics is translated to system level metrics, e.g., requests per second. *Quantitative metrics* use the workload history to predict the future workload. For instance, the workload of each hour of the day can be predicted taking into account the workload of previous days at the same hour. The workload of each week day can be calculated using the same week day of previous weeks.

Three quantitative techniques to estimate workloads can be highlighted [26]:

1. Regression methods,
2. Moving average, and
3. Exponential smoothing

Regression methods estimate a variable value as a function of other variables. Given a set of actual data points that measure the independent and dependent variables, the function that best approximates these values is searched. Linear regression searches for the linear function that minimizes the sum of square errors of each sample. This method can be used to translate business metrics into system metrics. For instance, the number of orders can be counted in each hour as well as the maximum number of requests per second in each hour. From that data set, linear regression can be used to obtain the relation between the number of orders and requests per second.

Moving average calculates a metric to forecast as the average of that same metric in n previous periods. For instance, the maximal number of requests per second for the next 10 minutes could be calculated as the average of the same metric in the three previous periods of 10 minutes. The method must be calibrated by defining the right duration and number of periods.

Exponential smoothing also takes the variable value in the previous periods to predict a new value, with the difference that a coefficient multiplies each value to decrease older values, and give more priority to last values. To compare approximations of different techniques and parameters, the Mean Square Error (MSE) can be used.

To summarize, capacity planning requires to know the workload in detail, the main applications, the type of transactions, and to use qualitative and quantitative metrics to forecast the future workload on multiple timescales.

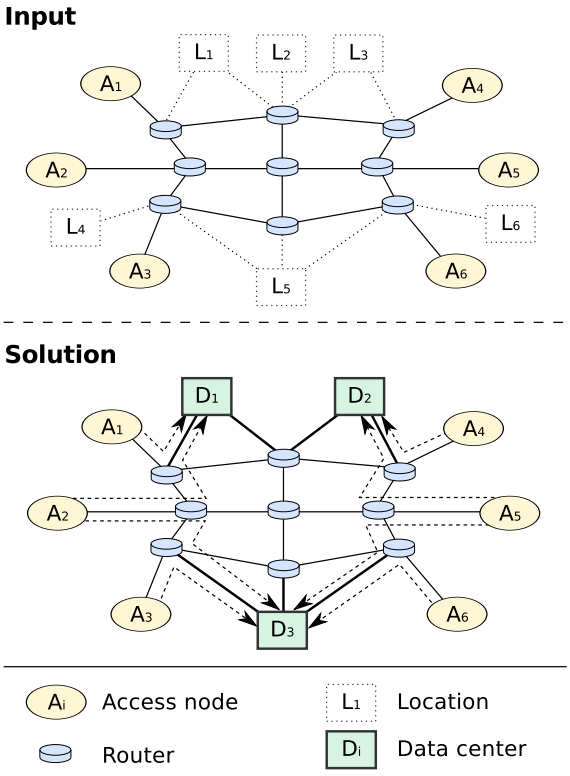

Figure 2.3 Data Center Location Problem (DCLP).

2.2 Data Center Location Problem (DCLP)

Data center location is the main decision to make in the Cloud Network Planning Problem (CNPP) that we proposed. In this section, we classify existing papers that address the DCLP including the articles that are part of this thesis.

QoS is a top priority because of the increasing use of cloud computing applications for personal and business tasks: to search for information, news, communication between people, social networks, entertainment. The delay between the user computers and the servers that host programs and information has a big impact on QoS. That delay is associated to the path composed of routers and links between a user computer and a data center. Thus, different data center locations produce different delays: when the data centers are closer to the users, the propagation delay is smaller and there are fewer intermediate routers adding queuing and transmission delay. Thus, cloud providers open data centers in multiple regions to locate the applications as close as possible to the users. For instance, the largest content delivery network Akamai has more than 1,000 small data centers around the world hosting images, videos, and applications for a large number of organizations [12].

In this context, an important problem to solve is where to locate the data centers to

minimize the delay experienced by user applications. Figure 2.3 shows an example. Users are aggregated as access nodes that demand services from the cloud. In this example, there are six access nodes. Very often, the choice of data center location is not totally open, but there are potential locations that are considered by the planner. In the figure example, there are six potential data center locations. Three of these locations were selected to place a data center and the service demands were routed from the access nodes to the data centers.

Delay is not the only issue impacted by the location of the data center. In fact, data center location has also an impact on costs such as the price of the land, electricity price, and the cost of environmental pollution that is produced by the type of energy—wind, solar, hydroelectric, nuclear, diesel, coal—that is available. For an effective data center location planning model, trade-offs between locations with low land costs, energy prices, CO_2 emissions and the proximity to the users should be used. In fact, from the cost minimization perspective, the optimal is to open a few data centers in cheap locations; from the delay minimization perspective, the best is to open many data centers near the users. On the other hand, to minimize CO_2 emissions the best solution is to open data centers only in locations where green energy is available.

This problem has an exponential number of possible solutions as a function of the potential locations, making necessary to use integer programming models, optimization solvers, and optimization techniques. In the following subsections we first present a review on a set of variants for this base problem. Finally, in the last subsection, we provide an analysis for further research on the topic.

2.2.1 Current approaches

Table 2.1 summarizes the most significant work relating to the DCLP. In the second column, the articles are classified by the type of objective to be minimized: communication delay, cost, energy consumption, pollution, social environment, and risks. The third column refers to the decision that can be reached when using the proposed model: data center location, assignment of access nodes to data centers, routing of traffic demands, and application location. In the final column, the different solutions approaches are portrayed: Mixed Integer Linear Programming (MILP) models solved with commercial solvers, simulated annealing heuristic [33], tabu search heuristic [13], Linear Programming (LP), and the classification method ELECTRE TRI [34].

Table 2.1 Data center location approaches.

Article	Objective	Decisions	Solution
Chang et al. [29]	- Delay	- DC location - AN to DC assig. - AN to backup DC	MILP solver
Goiri et al. [14]	- Cost	- DC location - AN to DC assig.	Simulated annealing + LP
Dong et al. [30]	- Energy	- DC location - AN to DC assig. - Routing	MILP solver
Larumbe and Sansò [31]	- Delay - Cost - Pollution	- DC location - AN to DC assig. - App. location - Routing	MILP solver
Larumbe and Sansò [23]	- Delay - Cost - Pollution	- DC location - AN to DC assig. - App. location - Routing - Link capacities	Tabu search
Covas et al. [32]	- Risk - Social - Cost - Pollution	- DC location	ELECTRE TRI

Delay minimization with backup coverage

The first variant to the DCLP is to minimize the average delay between access nodes and data centers subject to a budget constraint and the assignment of backup data centers. A data center can fail for many reasons: energy outage, link cuts, router failures, remote attacks, software errors. Redundancy such as multiple network links and energy lines diminish the probability of failure and increases the availability. The probability of natural disasters must also be considered. One way to protect against this is to replicate cloud services in multiple data centers and have mechanisms to route the service demands to a backup data center when the primary fails. Chang et al. [29] proposed an integer programming model where each access node must be assigned to a primary data center and a backup data center. The study also considered the load of each data center, defined as the sum of the demand that it serves. The maximal load in a solution is the load of the data center with the largest demand. The maximal load was added to the objective function that is minimized. The lowest value that the maximal load can take is when all the loads are the same. That is why that strategy tends to balance the load between all the selected data centers.

Cost minimization

Another variant to this problem is to minimize the total network cost subject to QoS constraints. In this case, the service demand of an access node can be assigned to a potential data center only if the delay between the access node and the data center is lower than a fixed parameter. Goiri et al. [14] proposed a model to solve this variant and studied potential locations in the US considering energy and land costs. In addition, the expected availability of each data center, i.e., its expected available time over total time, was also considered. The combination of data centers in the solution must satisfy a minimum availability requirement for the entire system. The authors proposed a simulated annealing heuristic [33] combined with a linear programming model, and demonstrated that the optimal placement of data centers can save capital expenditures (CAPEX) and operational expenditures (OPEX).

Energy consumption minimization

Network elements such as IP routers and optical switches consume energy. Different data center locations imply different routing and thus a different amount of energy consumed by the network. Dong et al. [30] studied the minimization of the optical and IP router power consumption as a function of the data center location using linear programming models. They also analyzed the efficiency of locating data centers close to renewable energy sources versus transporting the renewable energy to the data centers. Since transporting energy over the grid provokes losses, there is an incentive to place data centers near energy sources. The cases analyzed presented reductions of 73% in consumed energy. Of course, this approach could increase the delay because data centers can be farther from the users than the delay optimization case, but locating data centers near green energy sources and cold climate is a strategy used by companies such as Facebook that recently built a data center based on hydroelectric energy in Sweden [35].

Delay, cost, and pollution minimization

There are different types of energy to power a data center—solar, wind, hydroelectric, nuclear, diesel, coal—and depending on the type, the CO_2 emissions will vary. For instance, hydroelectric power produces 10 g of CO_2 per KWh and coal produces 960 g of CO_2 per KWh [11]. Thus, a data center with 60,000 servers located in a place where hydroelectric power is available will produce much less pollution than one with the same number of servers but powered with coal. The DCLP oriented towards pollution reduction should minimize the CO_2 emissions of the whole network.

All these objectives—delay, cost, pollution—should be taken into account by the planners to analyze the DCLP from multiple perspectives. One way to address this is to embed all objectives into a multi-criteria function having one term per aspect, normalized in monetary terms. Thus, the delay penalty represents the cost of delay increase, for instance, the loss in revenues because of a worsening QoS. Pollution costs may come from regulatory penalties or loss in company image for using dirty energy. Finally, there are the traditional OPEX and CAPEX costs. In order to have an overall view on the trade-offs, planners can adjust penalties to analyze solutions in a comprehensive way.

Larumbe and Sansò [31] solved this problem through a MILP model that minimizes a multi-objective function composed of a communication delay penalty, the data center CAPEX, the data center OPEX, the server cost, the energy cost and a pollution penalty. Larumbe and Sansò [23] developed a tabu search heuristic for that problem. In these articles, the data center location was simultaneously solved with the placement of distributed applications, making the problem more general and flexible. The application layer is modeled as a graph of software components that exchange traffic. In this case, the decisions include which data centers to use in addition to which data center will host each software component.

Risk, social, economic and environmental criteria

Another approach to decide a data center location is to classify every potential location in multiple independent dimensions. Instead of assigning a unique monetary value to each location, a category Excellent (C1), Very good (C2), Acceptable (C3), or Bad (C4) is assigned to each criterion. The criteria to analyze include:

- *Risk:* flooding, earthquakes, fire, nuclear, crime.
- *Social:* life quality, life cost, skilled labor.
- *Economic:* investment costs, operational costs, attractiveness to customers.
- *Environmental:* renewable energy, free cooling, reuse of waste heat.

For instance, a location with no flooding risks, earthquakes, and with low criminality may be classified as C1 in the Risk criteria. However, it can be classified as C4 in the economic criteria due to high investment and operational costs. Covas et al. [32] used Multi-Criteria Decision Analysis (MCDA) techniques [34] to evaluate possible locations of a data center for Portugal Telecom. This approach has the advantage of not combining multiple criteria in a single monetary value, allowing the analysts the evaluation of the solutions. The disadvantage is that the delay between the final users and the data centers was not taken into account.

2.2.2 Analysis

In Subsection 2.2.1, we have seen that the DCLP depends on multiple aspects such as land cost, energy cost, availability of renewable energies, and proximity to the users. The preference of one of the approaches over another will depend at first on the objective the planners want to achieve. A group of articles [29, 14, 30] optimized a single one: delay, cost, or energy. Other articles [31, 23, 32] proposed the optimization of multiple objectives simultaneously.

Data center location is connected to the assignment of access nodes to data centers because each access node aggregates a set of users that will access the cloud services. Besides the assignment decisions, Larumbe and Sansò [31], Larumbe and Sansò [23], and Dong et al. [30] also defined the routing of service demands between the access nodes and the data centers.

When distributed applications are executed on the data centers, the DCLP should include the traffic demands between the software components. Traffic between software components will be traffic between the data centers. Thus, the communication delay should also take that traffic into account.

Regarding the solution methods, Mixed Integer Linear Programming (MILP) models are very powerful in expressiveness and in the possibility of using the state of the art solvers, but they typically have a limit in the size of instances to solve. For large networks, custom solution methods both exact and heuristics are needed.

Finally let us remind the reader that the DCLP is closely related to the classical problems of Facility Location that were very well studied and for which a large number of techniques have been proposed [36]. An avenue of future research is to leverage these techniques for the particular case of data center location.

Our contributions to the area of data center location that are presented in Chapter 3 include:

1. A comprehensive MILP model for the Cloud Network Planning Problem (CNPP) to simultaneously solve data center and software component location as well as defining routing and link capacities.

2. Flexible multi-criteria approach to optimize CAPEX, OPEX, QoS, power consumption, and CO_2 emissions.

3. Flexible representation of distributed cloud applications through a graph of software components.

4. Special consideration of the communication traffic between access nodes and data centers, and among the data centers themselves.

5. Minimization of communication delay to improve QoS.

6. Minimization of CO_2 emissions by choosing regions where green energy is available.

7. A tabu search heuristic that finds near optimal solutions and solve cases with 1000 potential data centers in less than 10 minutes.

2.3 Dynamic scaling

The problem of dynamically determining the number of servers required for each application is an important part of resource management in cloud systems. Our models require that information as an input, thus we analyze existing approaches in the literature and industry. Then, Section 2.4 analyzes more complex models that consider multiple data centers.

There is a particular type of software architecture that requires special treatment: applications employing server pools that answer similar requests. An example is a web server cluster that answers HTTP requests. This architecture has a software component that is replicated in a set of servers, and a *scheduler* is used to distribute the requests among the servers, as shown in Figure 2.4. The scheduling algorithm can be round-robin, weighted round-robin, random choice, or an alternative method accounting for the server load, response time, and locality of the information. The objective is to spread the load for a short response time, and the method is therefore called *load balancing*. However, the goal of low server load contradicts the objective of maintaining a high server resource utilization to reduce the number of servers and total energy consumption. The *dynamic scaling* mechanism, also known as auto scaling or elastic load balancing, aims at finding the minimal number of servers necessary to maintain the QoS specified by the SLAs. The adjectives "autonomic", "dynamic", and "elastic" indicate that the set of active servers answering requests can grow or shrink depending on the workload volume. During periods of the day when there are more user requests, the number of active servers in the cluster is increased. When the number of requests is low, the number of active servers is decreased, and the inactive servers are placed in sleep mode to save energy. If unexpected peaks in traffic occur, then the number of active servers is increased, and requests are still met with a high QoS. When different applications in a data center have traffic spikes at different times, the dynamic scaling algorithm uses a shared pool of available servers. This technique reduces the total number of servers relative to that required in the case of over-provisioning for each application peak.

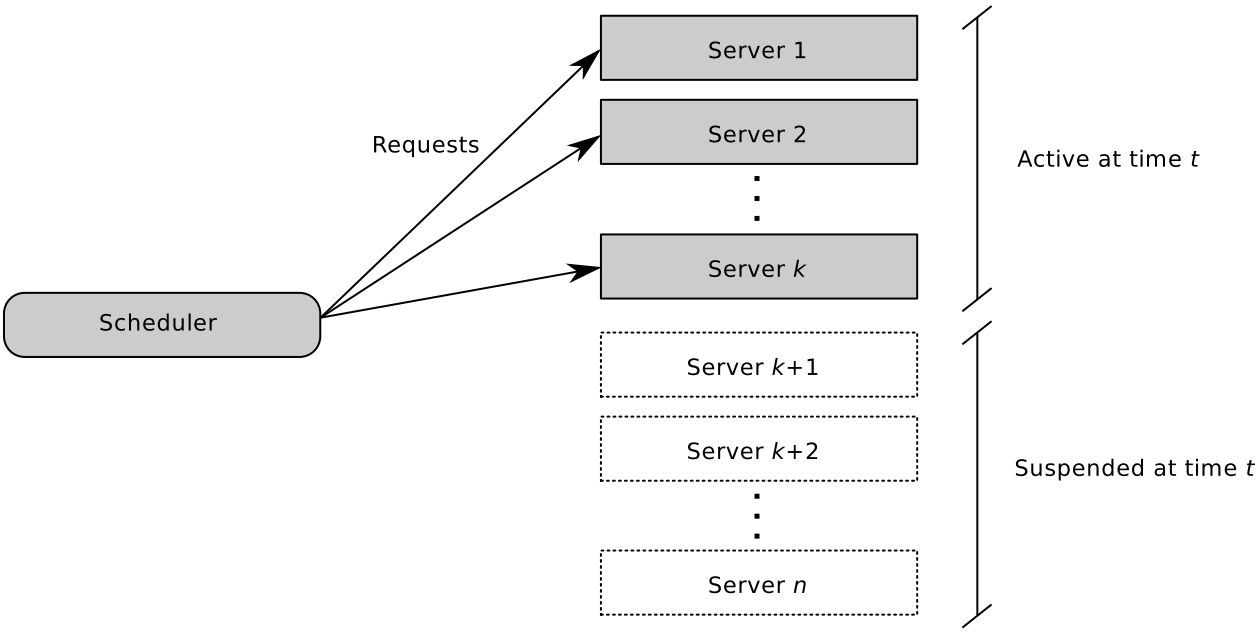

Figure 2.4 Dynamic scaling.

Table 2.2 Dynamic scaling approaches.

Article	Objective	Features	Dynamics	Solution
Pinheiro et al. [37]	- Server number	- Degradation threshold - Stability period - CPU usage as QoS degradation	- Reactive	- Add or remove one server
Urgaonkar et al. [38]	- Server number	- Multi-tier applications	- Reactive - Predictive	- G/G/1 estimation
Amazon [39] OpenStack [40]	- VM number	- Server resource metrics	- Reactive - Scheduled	- Add/remove n VMs - Add/remove x% VMs
Roy et al. [41]	- SLA infractions - VM cost - Config. cost		- Predictive	- Estimate workload - Calculate response time - Look-ahead

2.3.1 Current approaches

Table 2.2 shows a summary of the articles analyzed in this section. The classification starts with the objective to optimize: number of servers, number of VMs, SLA infractions, VM cost and configuration cost. The third column of the table presents particular considerations of each model.

The method to trigger a change in the cluster capacity may be reactive or predictive. In the reactive case, the triggering is based on the instantaneous values of the workload volume. In the predictive case, the triggering is based on the historical values and a prediction of the upcoming workload variation and required resources. Reactive methods are used on scales of minutes to respond to unexpected increases in the number of requests. Predictive algorithms are executed on scales of hours and days. Solution methods go from basically adding and removing VMs to more complex workload prediction strategies.

Server number minimization

We mention the pioneering work of Pinheiro et al. [37] that proposed a reactive algorithm using a degradation threshold and stability period. The *service degradation* can be defined, for instance, as the CPU utilization of the most busy server. The *degradation threshold* can be defined as 90% CPU utilization. The *stability period* is a time length where the system remains with a fixed number of servers. Given the set of active servers that are currently responding to requests, if the current service degradation exceeds the threshold value and the last resize operation preceded the stability period, then one server is added to the pool. Otherwise, if removing a server will keep the service degradation below the threshold and the last resize operation preceded the stability period, then one server is removed. This simple algorithm allows the cluster to grow or shrink one server at a time depending on the workload variation. The algorithm was applied to a web server pool with the maximum CPU utilization set to 90%, and servers were added and removed to keep the utilization below the threshold.

Urgaonkar et al. [38] solved the dynamic provisioning problem for multi-tier applications, which are applications consisting of multiple software components, each with a specific purpose. For instance, an application may include a web service, application server, and database. Each software component is replicated in a set of servers, e.g., a set of web servers, a set of application servers, and a set of database servers. The algorithm returns the number of servers that should be allocated to each set as output. It treats each server as a queue, and the minimal number of servers required to satisfy the SLA is calculated based on the server capacity. This algorithm is executed in both a reactive and a predictive fashion. The

predictive component anticipates the maximum request rate for the next hour based on the workload during the corresponding hour on previous days. The reactive method manages unexpected spikes that drive the workload volume above the predicted value. The request rate over the last few minutes is compared to the predicted value, and if the difference exceeds a specified threshold, then the number of servers is calculated using the observed request rate. The technique proposed by Urgaonkar et al. [38] was tested on a Xen based cluster with 40 machines and could handle sudden traffic spikes doubling the capacity in 5 minutes.

VM number minimization

From a cloud user perspective, the objective is to minimize the VM cost, which is proportional to the number of active VMs in the cloud, while satisfying the application QoS. Amazon provides an auto scaling method through Amazon Elastic Cloud Computing (EC2) [39].

Users can define triggers and policies for resource expansion and shrinking. A trigger is defined by a server metric to watch such as processing power and RAM, an aggregation operation such as the average over the whole cluster, an evaluation period, a number of periods, and a threshold. When the metric is above (or below) the threshold in a number of successive periods, then the scaling policy is triggered. The policy can add or remove a number of servers, or multiply the number of servers by a number, e.g., $+50\%$ or -15%. This method is able to perform cluster capacity variations when the workload changes progressively. It is also adapted to sudden traffic spikes, but starting a new VM could take some minutes.

Besides the reactive scaling, Amazon EC2 also provides *scheduled scaling*. In this scheme, the number of VMs needed is defined in advance. The capacity change can be defined for a specific time in the future, e.g., because an advertisement would produce a traffic increase, or can be periodic if the user knows the daily or weekly workload. For instance, the number of VMs could be doubled every day between 7 pm and 9 pm, and decreased by half during the weekends.

OpenStack is an open source Infrastructure as a Service (IaaS) system originated by RackSpace Hosting and the National Aeronautics and Space Administration (NASA) under Apache license [42, 43]. The project is supported by more than 150 companies including AMD, Intel, Red Hat, Dell, HP IBM, and VMware. OpenStack provides auto scaling through a project code named Heat with the same features as Amazon EC2 [40].

SLA infractions, VM, and configuration cost minimization

Roy et al. [41] determined the number of VMs that minimizes the total cost using a look-ahead method. The cost includes the number of SLAs that are not respected, the VM cost, and a

configuration cost that is proportional to the number of changes (added or removed VMs). This model predicts the workload over a given period using a formula that extrapolates the workload from the previous three periods. The formula is used to calculate the workload for the next N periods, beginning from the current time. In each of the next N periods, the algorithm will make a decision to increase, reduce or maintain the number of servers in a cluster. The look-ahead method generates a tree containing all possible decisions for the next N periods. For each tree branch, the net cost is calculated, including SLA penalties, VM cost, and configuration cost. The branch with the smallest cost is then chosen, and the algorithm adopts the first decision that generated the branch. This procedure is repeated periodically to adjust the resources to the demand at each time.

2.3.2 Analysis

The methods described in this section demonstrate that the application workload can be predicted and that the resources can be adjusted accordingly. We see an important gap between the methods proposed in the literature and the cloud provider solutions. While literature proposes predictive methods, current clouds only provide reactive methods. In any case, thanks to virtualization, VMs can be created and removed on the fly and predictive methods can be integrated by the cloud users.

It is worth noting that sometimes dynamic scaling is presented as a magical solution where service providers no longer need to define the number of VMs; in fact, that is not the case. For a correct application of dynamic scaling, the workload must be analyzed over time and the metrics that define it should be identified. A good predictive model adjusted for the analyzed workload should define the right amount of resources needed. The choice of a predictive model should also consider under what conditions the model remains valid. Security is an important aspect to take into account to verify that the workload is genuine. Otherwise, a malicious user could attack an application by introducing high levels of traffic, triggering scale up actions and entailing economic damages.

The dynamic scaling problem here presented optimizes the number of VMs in a single data center. With multiple data centers with prices that change over time, a coordinated dynamic scaling could reduce costs and improve the QoS. In that case, the number of VMs of multiple clusters in multiple data centers must be defined. The next section analyzes that matter and presents our contributions to this area.

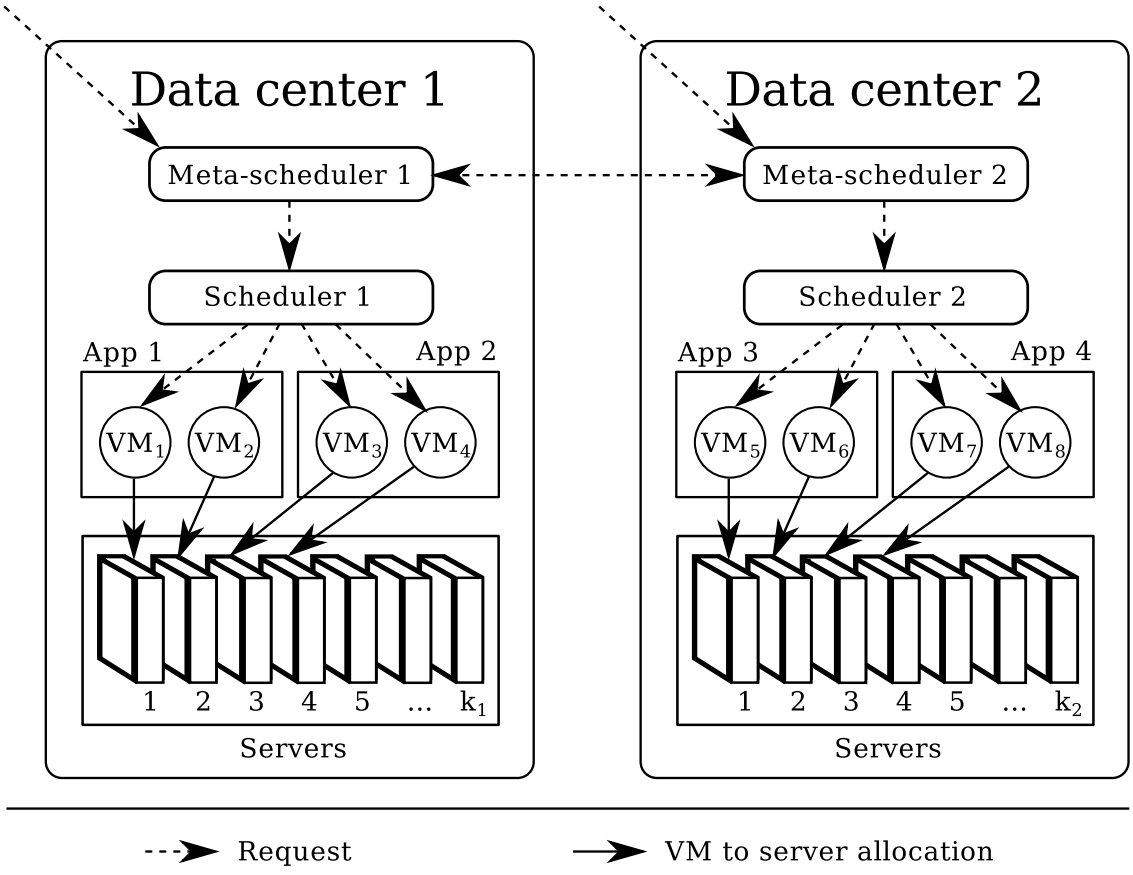

Figure 2.5 Meta-scheduling architecture.

2.4 Meta-scheduling: resource allocation across multiple data centers

The applications that process big amounts of data or operations handle scalability by executing parallel tasks in multiple servers. A *scheduler* receives requests and distributes them among a set of available servers. For instance, a massive website executes multiple web servers in parallel and the scheduler, in this case called load balancer, distributes the workload to achieve a short response time. In the case of HPC applications such as Hadoop [44], the information is split in multiple servers and each server solves a part of the task.

In a second level, the architecture can count on multiple data centers, each one with a set of servers executing tasks in parallel. In this case, a *meta-scheduler* distributes the workload among the data centers [45]. Figure 2.5 shows how requests are distributed by meta-schedulers and schedulers across multiple data centers. When a data center receives a high number of requests, servers queues grow, and the response time increases. Then, the meta-scheduler can redirect a fraction of the workload to other data centers with a lower load, and thus the response time decreases. Furthermore, this architecture can improve system availability because if a data center fails, its workload can be redirected to another data center. Counting on multiple distributed data centers also allows users to connect to the one with smallest delay, and then improve the QoS.

In this context, adding virtualization improves the resource usage among multiple applications. Each application has a set of VMs that process requests. VMs can be started and stopped in a dynamic way. At the time of starting a VM, a physical server is allocated to it. When a VM is stopped, the server resources are free for other VMs. Because each VM is able to process a maximal number of requests per time unit with the expected QoS, the number of VMs needed for an application will depend on the size of the workload at each time. Then, the number of servers allocated to an application varies as the workload size changes, and overprovisioning is avoided. If applications have workloads with high utilization in different periods of the day, fewer servers are used than reserving servers in a static way.

Note that the process of meta-scheduling can be done independently from the scheduling of each data center. Thus, this architecture can be used to manage a hybrid cloud, that is, a private data center in combination with an external provider such as Amazon EC2. Furthermore, multiple provider data centers can be used to increase availability and place VMs close to users. Because providers charge computing power as a service, the cost is a critical variable to minimize.

There are several approaches to optimize the meta-scheduling process. For each application, they all compute how many requests are handled by each data center. In the cases that virtualization is used, the result is how many VMs of each application are placed in each

data center. The following is a set of papers that address variants of the meta-scheduling problem across multiple data centers.

2.4.1 Current approaches

Table 2.3 presents the papers on meta-scheduling that are closest to our work, and also our paper on dynamic VM placement across multiple data centers. The second column of the table shows the objective to optimize in each work: QoS, cost, power consumption, and CO_2 emissions. The third column presents decisions and features that define each variant. The fourth column classifies variants depending on whether the optimization is solved online, for new arriving applications and requests, or offline, knowing the applications to be deployed in advance. In the same column, demands are static when resources allocated for each request remain constant over time. They are dynamic when the same demand has different values during the day, e.g., the number of VMs of an application changes over time. Some papers also considered providers that change prices over time. The last column shows the solution approach.

The following subsections describe the problem variants grouped by the objective.

QoS maximization

Leal et al. [17] proposed methods to schedule tasks for grid computing, the scientific predecessor of cloud computing. The grid is typically used to execute HPC jobs that process a big amount of data. For instance, the physical experiment Large Hadron Collider (LHC) generates data about collisions between particles. The HPC jobs are executed on multiple computer clusters managed by independent organizations. Each organization executes a meta-scheduler that sends jobs to the local cluster and to external clusters. For each job, the meta-scheduler must decide in which cluster it will be executed. Because each cluster has a computing capacity and the workload varies over time, the average execution time also changes. The objective of the proposed methods was to reduce the execution time of the HPC tasks. For that purpose, the average execution time is regularly calculated for each cluster, and the jobs are sent to the cluster with the smallest value. A further improvement was to schedule jobs in advance to reduce the job start time. Two connected grids with 2013 processing elements were simulated using workloads from the LHC.

Tordsson et al. [19] proposed an offline integer programming model to place VMs across multiple providers and maximize the QoS. Each VM type in each cloud has a capacity, e.g., in requests per second. The model chooses a data center and an instance type for each VM in order to maximize the capacity subject to system and budget constraints. The meta-

Table 2.3 Meta-scheduling approaches.

Article	Objective	Features	Dynamics	Solution
Leal et al. [17]	- Job exec. time	- High Performance Computing (HPC) task scheduling	- Online - Static demands	- Calculate node performance - Advance scheduling
Tordsson et al. [19]	- QoS	- VM placement - VM type - Budget constraint	- Offline - Static demands	- MILP solver
Chaisiri et al. [16]	- Cost	- VM placement - Reserved and on-demand plans - VM aggregation	- Online - Dynamic prices	- Stochastic prog.
Ardagna et al. [18]	- Cost - Response time	- VM placement - Capacity allocation - Load redirection - VM aggregation	- Online - Dynamic demands - Dynamic prices	- Non-linear optimization
Franceschelli et al. [46]	- Cost - Response time	- VM placement - Scaling	- Offline - Dynamic demands	- Palladio simulator
Kantarci et al. [47]	- Energy	- VM placement - Backbone virtualization - Comm. user-DC	- Semi-offline - Dynamic demands	- MILP solver
Larumbe and Sansò [24]	- Cost - Comm. delay - Power - CO_2	- VM placement - Comm. VM-VM and user-VM - Routing - VM aggregation	- Online - Dynamic demands - Dynamic prices	- Online multi-period heuristic

scheduling algorithm was tested with a HPC application across three cloud data centers and showed that using multiple cloud data centers improves performance and cost.

Cost minimization

Chaisiri et al. [16] proposed stochastic integer programming models for placing VMs across multiple cloud providers. The objective was to take advantage of the price difference between reserved plans and on demand plans. Reserved plans are cheaper but require to be acquired in advance, and the price of on-demand plan varies. The stochastic model handles uncertainty on future demands and prices. Tests based on HPC workloads of three clusters show that budgets can be minimized.

Cost and response time minimization

Ardagna et al. [18] solved the problem of capacity allocation of multiple cloud data centers and load redirection. In this case, the main decision to make is how many VMs should be deployed in each data center every hour to minimize cost while guaranteeing that the average response time of each application is below a threshold. Once the number of VMs in each data center has been determined for the next hour, the requests received by a data center can be processed locally or can be redirected to other data centers. When sudden spikes occur, a fraction of the workload can be redirected to decrease the response time. The models were solved through non-linear optimization solvers and closed formulas. The approach was evaluated with different parameters and workloads, and was also validated with a prototype environment in Amazon EC2.

Franceschelli et al. [46] proposed a tool for performance and cost evaluation of cloud systems. Cloud computing elements such as cloud providers, VMs, resources, and dynamic prices were modeled to calculate costs and QoS of multiple cloud providers. The proposed tool handles cloud virtual resources as an extension of Palladio, a program that predicts the QoS of applications [48]. The authors compared predicted values of QoS to an execution of the SPECWeb2005 industry benchmark [49]. Results reported that the model is precise for light load and conservative for high load. In a second analysis, the services of two real cloud providers—Amazon and Flexicale—were compared. Amazon reported response times between 12% and 20% lower than Flexiscale with 53% higher costs in the VMs used.

Energy consumption minimization

Kantarci et al. [47] solved the virtual machine placement on a network with multiple cloud data centers through a semi-offline MILP model with dynamic demands. In this case, the

backbone network that connects data centers and users was considered at the optical and virtual layers. The placement problem was simultaneously solved with the definition of the virtual topology and routing to minimize the energy consumption of the backbone network and the data centers. The scheme showed important improvements in the power consumption and resource utilization.

Cost, communication delay, power consumption, and CO_2 minimization

Our approach to distribute VMs across multiple data centers extends the meta-scheduling architecture to a graph of software components [24]. That graph represents complex applications such as multi-tier where a software component receives requests and then triggers requests to other software components. In this schema, an application is represented as a graph where each node is a software component, and each arc (i, j) represents the fact that software component i sends information to software component j. Each software component may execute a scheduler and process requests in parallel through multiple VMs.

Figure 2.6 shows an example of a social network inspired on Facebook's architecture [50]. In this case there are several types of software components: User Interface (UI), Web Service (WS), Memcache (MC), and Database (DB). The UI is executed on web browsers to handle user interaction and to generate HTTP requests. The WS receives HTTP requests and processes them through multiple VMs in parallel. The scheduling mechanism of the WS is a load balancer that distributes requests to minimize the response time. The WS gets information from the DB (from hard disks) and MC (from RAM). The DB is split in multiple servers to handle information of million of users. In this case, a hashing mechanism is used to determine which instance has the information required. MC is a software component that stores information in RAM, having much shorter response time than the DB. Multiple MC servers can be used to store a large number of objects in RAM. A hashing method is used to decide which MC server must store each object.

The solution of the meta-scheduler is which data center will host each software component, and thus its VMs. The decision to include users in the architecture through the UI allows to consider the communication delay between users and the WS because it impacts on the response time and thus in the QoS. Then, the VM placement is linked to the proximity of users as well as the data center capacities. When the model is applied to place software components across multiple cloud providers, multiple criteria are considered: VM prices, the price to send information, the energy efficiency of data centers, and CO_2 emissions depending on the type of energy available in each data center—e.g., hydroelectric, nuclear, diesel, coal—.

The proposed meta-scheduler [24] is online because existing applications and residual data center capacities are taken into account. Furthermore, the meta-scheduler is dynamic

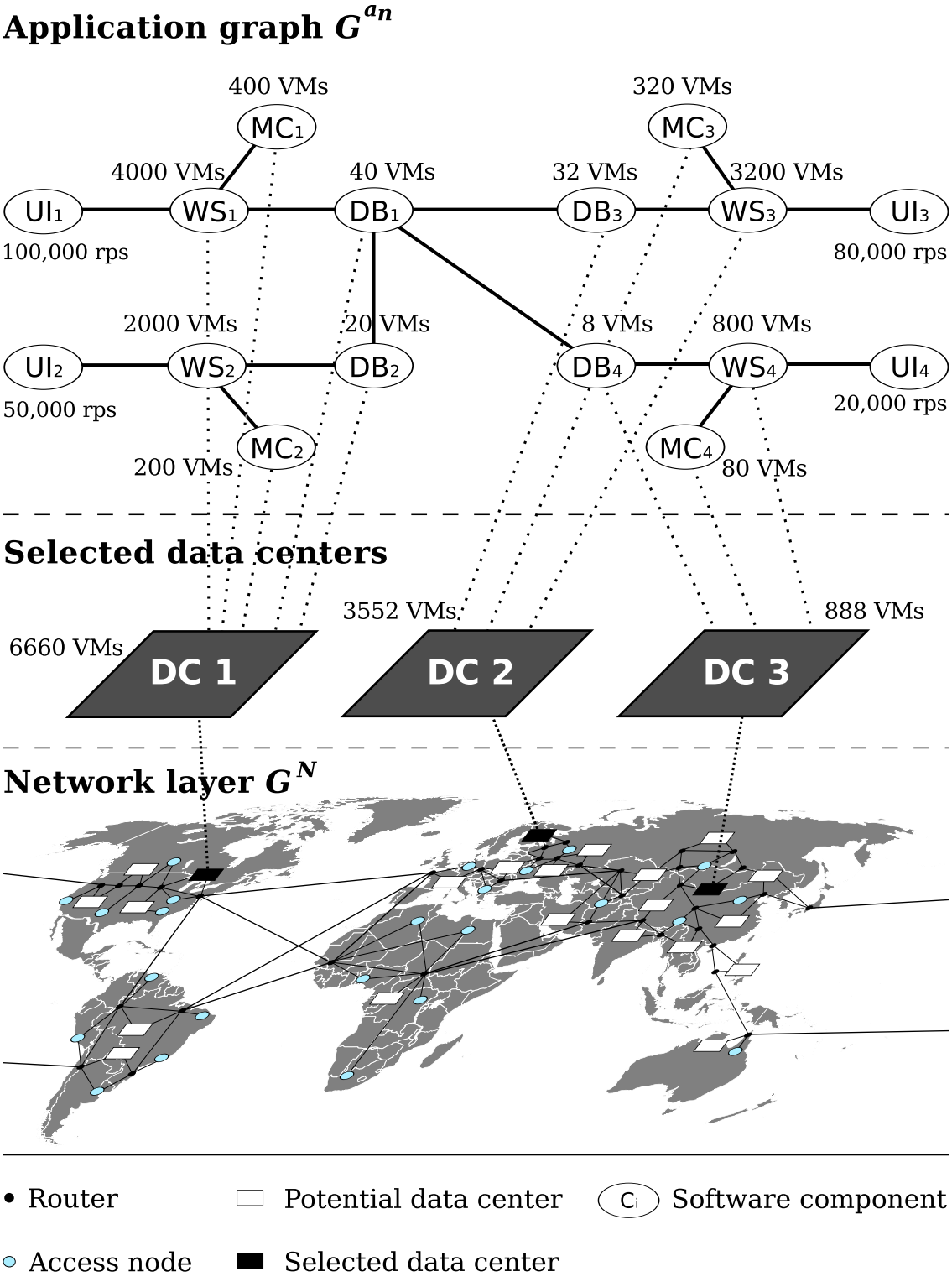

Figure 2.6 Application and network mapping.

because the workload of each period of the day is considered. In periods of high utilization, software components require more VMs than in periods of low utilization. Tests with more than 1000 nodes and realistic values for all the parameters were done. Results show how the cost is minimized, and that adding constraints decreases the response time up to 6 times, 30% of power consumption can be saved, and the CO_2 emissions can be reduced by a factor of 60.

2.4.2 Analysis

The meta-scheduling mechanism was studied in several articles because it comes from the history of grid computing [45]. In fact, that was the original objective of the first computer networks: to share computing resources among multiple computers [5]. New meta-scheduling approaches introduce virtualization technique. Thus, each application can be allocated to the VMs required at each instant and make a better resource usage. Fewer servers are needed for all the applications, and even unneeded servers can be suspended in periods of low data center workload.

More precisely, some papers grouped VMs into classes. A VM class, or software component in our approach, contains all the VMs that perform a same task in parallel. Then, defining the number of VMs of that class determines its computing capacity. The application workload can be matched by changing the number of VMs in its class. Chaisiri et al. [16], Ardagna et al. [18], and Larumbe and Sansò [24] considered VM aggregation in that way.

Given that academic research is mostly related to grid computing, multiple articles [17, 19, 16] were based on HPC workloads, and focused on data processing and not in the communication between VMs and users. Ardagna et al. [18] and Franceschelli et al. [46] considered standard web workloads from the industry. Larumbe and Sansò [24] and Kantarci et al. [47] took the communication delay between users and data centers into account, a fundamental aspect for the QoS of geographically distributed applications. Our approach emphasizes this aspect to get VMs close to users. In a similar way, Content Delivery Networks (CDNs) solve this by using a large number of data centers on the network edge [12].

The main objectives addressed in the literature are cost and response time minimization. There are articles that also minimized power consumption [47] and CO_2 emissions [24], increasingly important aspects for the environmental impact.

Finally, because starting a new VM can take in the order of minutes, the number of VMs in a class cannot be changed instantaneously. Sudden spikes in the workload could increase response time. That can be solved by redirecting a fraction of the spike requests to data centers with more available capacity in that period. Ardagna et al. [18] implemented a redirection mechanism to solve this issue. Leal et al. [17] consider the current average

response time of each data center at the time of scheduling each request. Our work considers dynamic capacities in number of VMs changing each hour, but does not implement request redirection. That feature can be added as an extension that is independent of the VM placement as in Ardagna et al. [18].

Comparing to the literature about meta-scheduling, the approach proposed in Chapter 4 presents the following contributions:

1. A flexible representation of applications as a graph of software components.
2. A MILP model that optimizes cost, QoS, power consumption, and CO_2 emissions.
3. Special consideration of communication delay between users and VMs, and among VMs themselves.
4. Online multi-period heuristic to solve the proposed model in real time.

2.5 Virtual machine placement within a data center

Virtualization is a technique that allows the optimization of resource allocation by reducing the number of servers used in each data center. To deploy a virtualization environment in a cloud data center, it is necessary to define the set of VMs to be placed on each server. In this section, we discuss the Virtual Machine Placement Problem and describe a set of studies that address it using different objective functions. Some of the studies focus on the reduction of investment and operation costs (or the number of servers), while others aim at reducing power consumption or maximizing QoS.

Virtualization allows multiple applications to be executed in a server without interfering with each other. These applications may belong to a unique organization in a private cloud or to different clients of a public cloud provider. Since the QoS of each application in a server should be stable, the VMs must be isolated and get a fair access to the server resources. The *hypervisor*, or VM monitor, is the program that allocates server resources to each VM guaranteeing the QoS. The hypervisor also provides security by isolating memory and storage for different VMs, and preventing guest operating systems from executing system-level instructions such as reboot the physical server.

The most popular hypervisors are Xen and VMware. Xen is a hypervisor started at the University of Cambridge Computer Laboratory and currently maintained as free software by the Xen community [15], and VMware is a commercial hypervisor [51]. Both of them can host VMs of different operating systems and are extensively used in the industry.

To grasp the virtualization potential for energy consumption reduction, take the following example. A server with 16 processing cores at 2 GHz with 4 VMs per core has the capacity

to execute 64 VMs at 500 MHz. If each VM is hosted on a dedicated server, 64 physical servers would be needed. If the VMs require a small portion of the server resources, this would be a waste of investment cost. It would also be a waste of energy because each server consumes a good amount of energy just being active. Furthermore, when some of the VMs in a processing core are idle, the others can use the whole processing power, thus a VM could reach 2 GHz even if it is sharing a processing core with other 3 VMs.

If every VM has the same amount of resources and all the servers have the same capacities, then each server can host a fixed number of VMs. In that case, the number of servers needed is proportional to the number of VMs and the best strategy to save energy is to consolidate the VMs in the smallest possible number of servers. Still, different applications require VMs with different capacities. Such cloud providers allow customers to choose among a set of predefined VM configurations. For instance, the cloud service Amazon EC2 has small VMs with 1 EC2 Compute Unit and 1.7 GB memory, medium VMs with 2 EC2 Compute Units and 3.75 GB memory, and large VMs with 4 EC2 Compute Units and 7.5 GB memory [39], where 1 EC2 Compute Unit is defined as the equivalent CPU capacity of a 1 GHz 2007 Xeon processor. Furthermore, a data center may have multiple server models with different resource capacities due to the hardware evolution over years.

Because of the heterogeneity in the VMs and the servers, we need to decide carefully which VMs to place in each server; this is known as the Virtual Machine Placement Problem. The goal is to minimize the number of servers while making sure that the resources required by the VMs hosted in each server are lower than the server capacity. The number of possible VM arrangements over the number of servers and VMs grows very fast, hence optimization techniques are needed to solve the problem.

2.5.1 Current approaches

Table 2.4 summarizes VM placement models analyzed in this section. As shown in the second column of the table, the papers are classified by the objective to minimize: the number of servers used, energy consumption, response time, migration cost, profit, and communication delay. The third column presents some particular features of each model such as VM migration, availability, traffic between VMs, traffic between VMs and users, and routing.

The algorithms are classified as offline, semi-offline, or online depending on how they are used. An offline algorithm takes the entire set of VMs to be placed and a set of servers as input and provides the optimal placement as the output. The offline algorithm is executed at specific times and places all of the VMs in a data center simultaneously. On the other hand, online algorithms receive VM requests over time and solve the placement problem for the new VMs considering the resources used by the VMs that are already in place. There

Table 2.4 Virtual machine placement approaches.

Article	Objective	Features	Dynamics	Solution
Speitkamp and Bichler [52]	- Server number		- Offline - Static and dynamic versions	- Branch and bound - Heuristics
Srikantaiah et al. [3]	- Energy	- Analyzed energy per request	- Offline - Static demands	- Heuristic
Beloglazov et al. [53]	- Energy	- Migration	- Online - Static demands	- Sort VMs by decreasing CPU requirement - Sort servers by energy efficiency
Verma et al. [54]	- Energy	- Avoid VMs interference - Migration	- Semi-offline - Dynamic demands	- Sort servers by energy efficiency
Addis et al. [55] Ardagna et al. [56]	- Response time - Migration cost - Profit	- Processor scheduling - DVFS - Migration - Availability	- Offline - Dynamic demands	- Local search
Addis et al. [57]	- Response time - Migration cost - Profit	- Processor scheduling - DVFS - Migration - Availability	- Offline - Dynamic demands	- Descentralized local search
Meng et al. [20]	- Comm. delay	- Traffic VM-VM	- Offline - Static demands	- Cluster algorithms
Fang et al. [22]	- Comm. delay - Energy	- Traffic VM-VM	- Offline - Static demands	- Cluster algorithm - Tabu search
Guo et al. [21]	- Comm. delay	- Traffic VM-VM - Routing	- Online - Dynamic demands	- Allocate cluster to VDC. - Allocate server to VM.

is an intermediate class of algorithms called semi-offline algorithms, which consist of offline algorithms executed in a periodic fashion, e.g., daily, weekly or monthly.

VM placement models are also classified based on whether they consider static or dynamic resource requirements. In the case of static algorithms, the VM requires a constant amount of resources. In the dynamic case, the resource requirements vary over time. For instance, in a static algorithm, one processor and 4 GB RAM may be allocated to the VM over its entire life cycle. In a dynamic algorithm, a specific VM may consume 100 MHz on average with 1 GHz peaks between 7 am and 10 pm and an average of 1 MHz with 20 MHz peaks between 10 pm and 7 am. Finally, multiple solution methods were proposed in the literature: branch and bound, local search, hierarchic local search, cluster algorithms, tabu search, and custom heuristics.

Cost minimization

The main objective of this problem is to reduce the cost of investment and operation of servers, including energy consumption, while assuring the required QoS. Each VM requires a specified quantity of each resource: CPU (in instructions per second), RAM (in bytes), network bandwidth (in bps), and storage (in bytes). Each server has a specified capacity for each of these resources. A feasible solution must respect the server capacities in consolidating the VMs on the servers. Speitkamp and Bichler [52] proposed an integer programming model for this problem and solved it using a branch and bound algorithm. They also adopted a heuristic algorithm known as *first-fit decreasing* from the bin-packing problem. This algorithm sorts the VMs in decreasing order of CPU consumption, and then one VM at a time is placed on the first server with sufficient capacity to host it. Speitkamp and Bichler [52] also proposed a model with dynamic demands, which was solved using a heuristic algorithm combining linear programming relaxation with an integer programming model.

Energy consumption minimization

There are also models that specifically minimize the quantity of energy consumed by the servers. Srikantaiah et al. [3] analyzed the impact of the server consolidation on the energy consumption and service request throughput. Reducing the number of active servers decreases the energy consumption per time unit. However, as shown in Figure 2.7, the energy consumption per service request displays a more complex behavior. When the server is too heavily loaded, the efficiency of the VMs and applications decreases. That is, each service request requires more time to be processed, thus total amount of energy per request also increases when the server is stressed. When the resource utilization is low, the applications

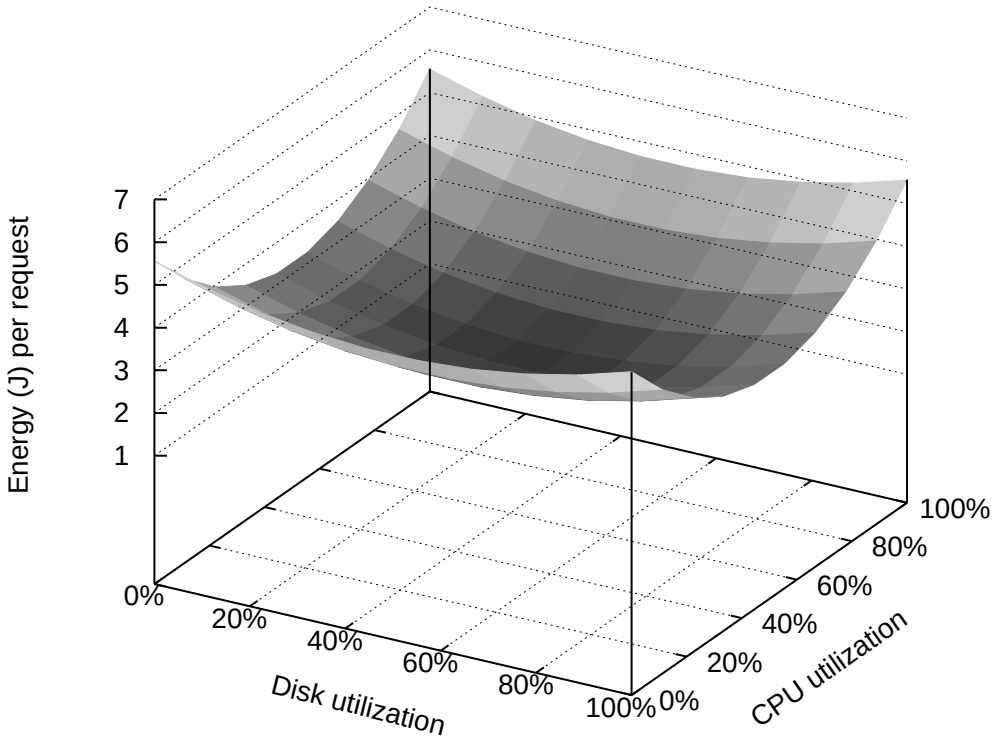

Figure 2.7 Energy consumption per service request. Adapted from Srikantaiah et al. [3].

are more efficient and the requests require less response time, but the energy consumed per request is high because the base energy required to keep a server on is divided among few server requests. The optimal value of the energy per service request was achieved with a CPU utilization of 70% and disk utilization of 50%. An offline heuristic model with static demands was developed to find the server resource levels that simultaneously optimize the QoS and energy consumption.

Beloglazov et al. [53] also performed an analysis of the server power consumption. In this case, a modified version of the best fit decreasing algorithm for bin-packing was used in an online algorithm with static demands. The VMs are sorted in decreasing order of CPU utilization, and for each VM, the server with the smallest increase in energy consumption is chosen. A new server is therefore turned on only if the active servers have insufficient capacity. In this algorithm, the servers with better energy efficiency are chosen first.

Verma et al. [54] studied the correlation between the resource utilization levels of various applications on the same machine over time, and a semi-offline placement algorithm was proposed to account for the dynamic demands. It is important to consolidate the VMs without impacting the application performance. To achieve this, the CPU utilization of each VM is monitored during a specified time interval. A placement algorithm is then used to group the VMs into clusters, with each cluster containing VMs with overlapping utilization peaks. The VMs in a given cluster could potentially have conflicts if placed on the same server. The servers are sorted by decreasing energy efficiency, and the VMs are placed on one server at a time. At each step, a subset of each cluster is chosen to load the server below its capacity. Experiments with real application traces demonstrated that the algorithm reduces the energy consumption with very few SLA violations.

Response time minimization and profit maximization

Another objective when specifying the VM locations is to minimize the response time or maximize its inverse, the throughput. Addis et al. [55] and Ardagna et al. [56] proposed an offline model with dynamic demands that determines the placement of the VMs, the frequency and voltage at which each CPU operates (DVFS), and the fraction of the CPU used for each VM. The objective function combines a linearly decreasing revenue over the response time, the cost of the servers as a function of frequency, the cost of activating new servers and the cost of moving VMs. The average availability of each server and the availability required for each request are also considered. The requests are divided into service classes requiring different degrees of availability. When the requests are served by VMs located on different servers, the availability increases. The solution approach in this model was a local search with movements related to each group of decision variables.

Addis et al. [57] proposed a hierarchical approach for this problem where the servers are split in clusters. The architecture is composed by a Central Manager (CM) and one Application Manager (AM) per each server cluster. The CM allocates one AM to each VM, and each AM solves part of the problem for the VMs and servers in that cluster. The AM defines the VM placement within the cluster, the load balancing, the capacity allocation to VMs running in each server, switching servers to active and sleep states, and increasing or decreasing each server CPU frequency. The proposed solution required orders of magnitude less execution time than the centralized approach and achieved similar values.

Communication delay minimization

Most of the VM placement models consider the CPU and disk requirements of the VMs, and some of them also consider the network bandwidth. It is better to consider the traffic between each pair of VMs rather than treating each VM independently. This way, the VMs with a high level of traffic are located nearby in the data center network.

Meng et al. [20] defined an offline model with static demands that takes a traffic matrix between the VMs and a delay matrix between the servers as input. The objective is to minimize the average delay of the traffic exchanged by the VMs. The method employs a cluster algorithm to partition the VMs and another cluster algorithm to partition the servers. The VMs that exchange more traffic are thereby grouped in the same cluster of VMs, and nearby servers in the network belong to the same cluster. The algorithm then maps each VM cluster to a server cluster and repeats the operation within each cluster in a recursive fashion. A similar approach was taken by Fang et al. [22] that used a cluster algorithm to group VMs, and then mapped each cluster to a server rack through the Quadratic Assignment Problem (QAP). The QAP was solved through a tabu search heuristic. In that case, the authors also proposed to minimize the power consumption by turning off network switches.

Guo et al. [21] proposed an online algorithm with dynamic demands. In this case, a Virtual Data Center (VDC) is defined as a set of VMs with a common purpose and an associated Service Level Agreement (SLA). Each VDC has a traffic matrix between the VMs. The network has a set of server clusters with different sizes. These clusters are defined depending on the number of hops between the servers and can overlap between them. For instance, a 2-hop cluster is a server rack, and a 4-hop cluster is a set of racks. Each VDC is assigned to one of the server clusters, and then each VM is assigned to one server. The criterion is to choose the smallest cluster with enough available servers and residual aggregated bandwidth between the servers. That way, the VMs are located in nearby servers, reducing the delay and guaranteeing the bandwidth reservation.

Migration of VMs

When a VM placement algorithm is executed, a set of VMs may already be running on the servers. In the online case, the VM requests arrive sequentially and the algorithm places the VMs with no knowledge regarding future VM requests, often yielding a placement that is far from optimal. Mechanisms for moving VMs between servers [58] help to solve this problem and improve the current allocation. A disadvantage of these methods is that a VM that is moved must be temporarily suspended and also generates traffic in the network; moving all of the VMs at the same time, therefore, is not a valid solution. Beloglazov et al. [53] resolved this issue in three steps. First, the servers whose resource utilization is above a specified threshold are detected. Then, for each host, the smallest subset of the VMs that can be moved to reduce the resource utilization to a level below the threshold is selected. Finally, each VM in the subset is moved to the server with the smallest increase in energy consumption.

2.5.2 Analysis

We have seen that the VMs should be consolidated to reduce the number of servers, cost, and energy consumption; that the response time should be reduced to optimize the QoS and provider revenues; and that the communication delay may have an important impact on the application performance.

Offline algorithms may be used for placing VMs for applications that are known to stay in the data centers for a long time. For the typical cloud case, online algorithms are preferable to handle new applications and remove VMs no longer needed. Considering static resource demands is acceptable for the case in which the application workloads remain constant and a fixed set of VMs is suitable for an application. However, when the workload presents fluctuations over time, dynamic demands should be considered to dynamically scale the number of VMs needed. VM migration is a feature that should be carefully analyzed for each particular case because moving VMs generates network traffic, suspends the execution during a time interval, and the network protocols such as TCP can suffer delay.

Virtualization intrinsically reduces the number of servers used and hence the energy consumption; a good VM placement algorithm can further improve cost and energy savings as well as improve the QoS. That is why there are many challenges to solve in this field. As more public clouds arise, cloud federation will become a reality [59] and VM placement algorithms should take that into account to locate VMs near the users reducing the delay and also saving energy costs. Traffic between VMs is also an important issue because big data applications produce a large amount of traffic within the data center. Thus, the data center

network should present topologies with a high degree of server connectivity and bandwidth, and VM placement models should consider that topology. Given the number of VMs and servers involved, and cluster based applications are the most promising architecture to handle big amounts of data, the multi-level approaches are good ways to place clusters and VMs.

Our contributions on the assignment of VMs to servers presented in Chapter 5 include:

1. A Mixed Integer Programming (MIP) model for online VM placement to optimize QoS and power consumption.

2. Special consideration for communication traffic among VMs to reduce communication delay and increase throughput among VMs.

3. The dynamic workload is considered to vary the number of VMs of each application.

4. Resources in each server rack are reserved to accommodate VMs that arrive in peak periods and further reduce delay.

5. A hierarchic heuristic that can be used to deploy, remove, and resize applications for a data center with more than 100,000 servers.

2.6 Conclusion

The ever-increasing number of available applications and data increments the number of servers and power consumption in the data centers. Cloud computing systems aim at reducing power consumption through optimal resource allocation. This chapter presented a set of problems regarding the location of cloud data centers and dynamic allocation of cloud computing resources, proposed solution methods, and our contributions.

The selection of data center locations to host cloud computing applications has an important impact on the response time, cost, and CO_2 emissions. It is therefore recommended that operators employ multi-criteria models to aid in the decision making process. Large networks can have more than 1,000 data centers, thus efficient algorithms are needed. Both exact and heuristic approaches are fundamental tools in the solution of these models. Chapter 3 presents a comprehensive approach for the planning of cloud computing networks.

In the everyday management of cloud systems, dynamic scaling allows applications to grow and shrink depending on the workload. This technique is responsible for the illusion of infinite resources in cloud computing paradigm, allowing the cloud data center to be seen as a computer. Proper dynamic provisioning can only be achieved with good reactive and predictive models that find the right amount of resources needed for each application. Analysts should carefully study application workloads to develop and/or setup these models.

Furthermore, metrics can be improved with a coordinated management of applications distributed across multiple data centers. The availability is higher because failures can be handled; response time is decreased by placing VMs close to users and by redirecting traffic to available data centers; power consumption is decreased by placing VMs in efficient data centers, and CO_2 emissions are minimized with data centers provided with green energy. Chapter 4 optimizes these aspects through a meta-scheduler that places VMs across multiple cloud providers.

Virtualization is the key cloud computing strategy to reduce the number of servers and power consumption. A right implementation of this technique includes a placement strategy that maps VMs to servers. This policy should take into account VM resource requirements and server configurations to guarantee that VMs are isolated and do not interfere with each other. In addition to the server resources, network topology and traffic are fundamental resources in VM placement policies because they impact the communication delay and the QoS of applications. Chapter 5 proposes a method to assign VMs to servers by considering communication among VMs and dynamic workload.

The location and allocation of cloud data center resources, servers, virtual machines, and software components is a challenging area of research for the development of a cloud computing paradigm that will be economically and ecologically sustainable.

CHAPTER 3

ARTICLE 1: A TABU SEARCH ALGORITHM FOR THE LOCATION OF DATA CENTERS AND SOFTWARE COMPONENTS IN GREEN CLOUD COMPUTING NETWORKS

Federico Larumbe and Brunilde Sansò

Accepted for publication in
IEEE Transactions on Cloud Computing
on June 2, 2013

Abstract

The ubiquity of cloud applications requires the meticulous design of cloud networks with high quality of service, low costs, and low CO_2 emissions. This paper presents a planning problem and an extremely efficient tabu search heuristic for optimizing the locations of cloud data centers and software components while simultaneously finding the information routing and network link capacities. The objectives are to optimize the network performance, the CO_2 emissions, the capital expenditures (CAPEX), and the operational expenditures (OPEX). The problem is modeled using a mixed-integer programming model and solved with both an optimization solver and a tabu search heuristic. A case study of a web search engine is presented to explain and optimize the different aspects, showing how planners can use the model to direct the optimization and find the best solutions. The efficiency of the tabu search algorithm is presented for networks with up to 500 access nodes and 1,000 potential data center locations distributed around the globe.

Keywords

Application component placement, cloud computing, cloud federation, data center location, energy efficiency, environmental impact, green networking, network planning, tabu search, virtual network embedding.

3.1 Introduction

With the evolution of the Internet, we are witnessing the birth of an increasing number of applications that rely on the network; what was previously executed on the user's computers as stand-alone programs has been redesigned to be executed on servers with permanent connections to the Internet, making the information available from any device that has network access. Instead of buying a copy of a program, users can now pay to obtain access to it through the network, which is one of the models of cloud computing [60], Software as a Service (SaaS). The continuous growth of Internet bandwidth has also given rise to new multimedia applications, such as social networks and video over the Internet; and to complete this new paradigm, mobile platforms provide the ubiquity of information that allows people to stay connected.

Service providers may own servers and data centers or, alternatively, may contract infrastructure providers that use economies of scale to offer access to servers as a service in the cloud computing model, i.e., Infrastructure as a Service (IaaS). As users become more dependent on cloud services and mobile platforms increase the ubiquity of the cloud, the quality of service becomes increasingly important. A fundamental metric that defines the quality of service is the delay of the information as it travels between the user computers and the servers, and between the servers themselves.

Along with the quality of service and the costs, the energy consumption and the CO_2 emissions are fundamental considerations in regard to planning cloud computing networks. Data centers consumed 1.5% of the energy in the U.S. in 2006, which is equivalent to the amount of energy consumed by 5.8 million households [61]. Each type of energy, such as wind, hydroelectric, nuclear, diesel, and coal, introduces a different amount of CO_2 into the environment, and each potential data center location will be provided by one or more of these energy sources. Thus, choosing data center locations supplied by green energy sources can greatly reduce environmental pollution.

In this study, we solve the problem of designing a cloud computing network by answering the following questions:

- What potential data centers should be used in the network?
- Which data center should host each of the software components of the cloud applications?
- How many servers will be hosted at each data center?
- How will the information be routed through the network?

- What are the link capacities required to carry that information?

All these questions are interrelated and have an impact on the quality of service, energy consumption, cost, and pollution. However, in practice, these questions are tackled separately and very often, in different time frames. In this paper we propose the simultaneous decision of data center location, software component placement, and other planning elements in order to provide a comprehensive optimization framework for future reference. Once the network is designed and in operation, complementary models can be used to manage the system in a dynamic way, and interesting research avenues are open in that direction [62, 63].

The criteria to choose the optimal solutions in our framework are embedded in a multi-objective function that allows planners to weight each attribute according to their priorities. The objective function is composed of the following metrics: traffic delay, energy consumption, CO_2 emissions, traffic cost, server cost, data center capital expenditures (CAPEX), and data center operational expenditures (OPEX).

The proposed problem is formalized as a Mixed Integer Linear Programming (MILP) model and solved, first with AMPL-Cplex, and then with a very efficient tabu search heuristic. That heuristic is strongly needed because the numerous integer variables of the model produce very high execution times in a general optimization solver. In fact, AMPL-Cplex took up to one hour to solve small instances of 24 potential data center locations. The cases used for this study were networks with up to 500 access nodes and 1,000 potential data center locations, which corresponds to the size of the largest cloud networks currently available [12]. We show that the tabu search algorithm presented in this paper achieved very small optimality gaps, and the execution time ranged from some milliseconds to less than 10 minutes.

The framework presented in this paper helps understand each aspect of a cloud network in a formal way, and multiple actors may be interested in having optimal solutions to this problem. One actor may be an infrastructure provider that needs to deploy or extend a data center network used by service providers. Another actor may be a service provider that needs to choose the cloud data centers to deploy global distributed applications and to decide the number of servers to host in each one of them. Yet another actor may be organizations with private clouds who want to solve the whole network design in an integrated and optimal way.

The remainder of this paper is structured as follows. Section 3.2 presents a literature review of related problems and models. Section 3.3 describes the problem and proposes a MILP model. Section 3.4 specifies the tabu search heuristic that was developed to solve large problems. Section 3.5 presents a case study of a search engine application that requires data centers and software components to be placed around the world. Section 3.6 shows optimal solutions for the case study and analyzes how the multiple objectives interact. This section

also shows the optimality gap of the tabu search heuristic and the execution time depending on the instance sizes. Finally, the conclusions are presented in Section 3.7.

3.2 Related work

The problems of facility location, such as capacitated facility location [67, 68, 69], originate in operations research, and extensive literature exists on them [36]. As surveyed by Larumbe and Sansò [63], some articles on data center location were recently presented [29, 14, 30, 32]. Table 3.1 summarizes the problems of data center location and provisioning closest to our work. Chang et al. [29] considered access nodes that aggregate user traffic, and each one of them must be assigned to a primary data center and to a backup data center. The objective of the proposed linear programming model was to minimize the delay between users and data centers. In Goiri et al. [14], a model that minimizes the total cost of traffic, servers, and data centers was presented. Potential locations in the U.S. were studied, considering energy and land costs. That approach is the closest to the model presented in this paper because cost, energy, delay, and CO_2 were also considered. The main difference is that our model takes into account distributed applications with traffic between user devices and servers and between the servers themselves, which has an impact on the routing and capacity assignment of the backbone network. Furthermore, our approach includes a multi-criteria objective function allowing planners to minimize delay, CO_2 emissions, as well as costs. Regarding the solution method, Goiri et al. [14] proposed a simulated annealing heuristic combined with a linear programming model. Dong et al. [30] focused on the minimization of optical and IP router power consumption as functions of the data center location through linear programming models. The proposed strategy was to place data centers close to energy sources to reduce power losses in the grid. Covas et al. [32] proposed a method to select potential data center locations by comparing multiple criteria, such as energy cost, security, availability of telecommunications networks and labor. The delay between users and data centers was not taken into account in that model.

With respect to the application location, Stone [64] was the first to model the software component placement considering the information traffic between the components. The proposed solution method based on a maximum flow algorithm was restricted to the case of two processors. Rao et al. [65] served web requests from multiple data centers to minimize the total energy cost by changing the workload assignment, depending on the energy price, and guaranteeing the quality of service. The problem was linearized and then reduced to a minimum cost flow problem. Buyya et al. [62] studied dynamic provisioning of applications from multiple cloud vendors and did a performance study with the CloudSim toolkit. That

Table 3.1 Data center location and provisioning approaches.

Article	Objective	Decisions	Solution
Chang et al. [29]	- Net. delay	- DC location - AN to DC assig. - AN to backup DC	MILP model + solver
Goiri et al. [14]	- Cost	- DC location - AN to DC assig.	Simulated anneal. + LP
Dong et al. [30]	- Energy	- DC location - AN to DC assig. - Routing	MILP model + solver
Larumbe and Sansò [31]	- Net. delay - Cost - Pollution	- DC location - AN to DC assig. - App. location - Link capacities	MILP model + solver
Covas et al. [32]	- Risk - Social - Cost - Pollution	- DC location	ELECTRE TRI
Stone [64]	- N. delay	- App. placement	Max. flow
Rao et al. [65]	- Energy cost	- Workload assig.	Minimum cost flow
Buyya et al. [62]	- Cost - Energy - Resp. time	- App. placement - Resource alloc. - Workload migration	CloudSim
Chowdhury et al. [66]	- Cost - Load balance	- Virtual network embedding	LP + multicom. flow

paper shows that the cloud federation model yields important benefits in quality of service and costs.

Because the application layer and the network layer are two separate graphs and because the problem includes mapping the application layer on top of the network layer, a related problem is virtual network embedding, which maps a virtual network with virtual routers and virtual links on top of a physical network. Each virtual node must be located on a physical node, and each virtual link must be routed through a physical path. Chowdhury et al. [66] proposed an integer programming model and solved it using linear relaxation and rounding techniques. With respect to defining the routing and the link capacities in multilevel networks, Plante and Sansò [70] presented a survey on the models and solution approaches for the network synthesis problem.

Regarding our previous work, Larumbe and Sansò [71] presented a model that simultaneously solved the location of data centers and software components with the routing. This model was an initial model that minimized the network delay but did not consider the energy and the pollution aspects. Larumbe and Sansò [31] solved a model that considered energy and pollution aspects through an AMPL-Cplex program by adding the search for optimal link capacities. The new aspect caused us to discard the assumption of Poisson traffic, and the delay of each link was taken from real measurements.

The present study includes the following original contributions:

1. A highly efficient tabu search heuristic to minimize delay, cost, energy, and CO_2 emissions in the jointly determined location of data centers and software components, routing, and link dimensions.

2. Calculation of the energy consumed by switches and routers, in addition to the servers.

3. Trade-off analysis between the multiple objectives.

3.3 Problem description

The problem is to define the location of data centers and software components, the number of servers in each data center, the information routing, and the capacity of each link in the network. The design objective is to minimize the network delay, the cost, the energy consumption, and the CO_2 emissions. This problem is formalized as an MILP model that minimizes a multi-criteria objective function.

The problem setting is an organization that offers cloud services to its users through a data network. The organization must optimize the network design to provide the best possible quality of service while reducing costs, energy consumption, and CO_2 emissions. The

applications that will be executed in the network are represented as a graph $G^A(V^A, A^A)$, where each node is a software component and each arc is a traffic demand from one software component to another. Each software component may be executed on a user computer or on a server. A single software component can even span multiple servers to achieve its purpose. The applications will be deployed on a computer network $G^N(V^N, A^N)$ composed of access nodes $\mathcal{A}$, backbone routers $\mathcal{B}$, existing data centers $\mathcal{D}$, and possible locations to open new data centers $\mathcal{L}$. The set $\overline{\mathcal{D}}$ denotes all the potential data centers, i.e., the union of $\mathcal{L}$ and $\mathcal{D}$. Each access node represents an aggregation of users, which may be an Internet Service Provider (ISP), a mobile company, a city, a region, or a country, depending on the case. The users will access the cloud services hosted in the data centers. The nodes in G^N are connected through the backbone links, typically by optical fiber. The link capacities, measured in number of channels, are decision variables. Because not every node is suitable to host every software component, P is the set of possible assignments of software components in V^A to network nodes in V^N.

Figure 3.1 shows an example of a web search engine that must be deployed on a data center network. The application is composed of user interfaces, web servers, and World Wide Web indexes. Each access node aggregates the users of a whole city. There is one user interface for each access node, representing the client component that is executed on the user computers of that access node. There is also a web service that answers requests to that access node. The same web service may require multiple servers to handle all the requests in an efficient way. The web service makes index queries to the web index where the associations between keywords and pages are stored. Each web index may also require multiple servers with multiple hard disks to store the entire index. In the example, WI_1 is the master index and all other indexes are replicas of it. In this case, the problem is to choose a subset of the potential data centers and locate each web service and each web index in one of the data centers. The following sections provide the parameters and decision variables of the problem.

3.3.1 Network traffic

The software components send and receive information between them, and this information is divided into packets that will be sent through network paths. Each step in a path adds processing, waiting, transmission and propagation delay. While the traffic in a link is under a maximal allowed utilization, the link delay is bound by a constant. For example, a specific link could present packet delays of less than 1 millisecond while its traffic is less than 70% of its capacity. In the same way, the delay incurred by the packets in a path is bound by the sum of the link delay bounds. The strategy in this problem is to locate software components that exchange traffic in close nodes in terms of delay such that the delay is minimized and

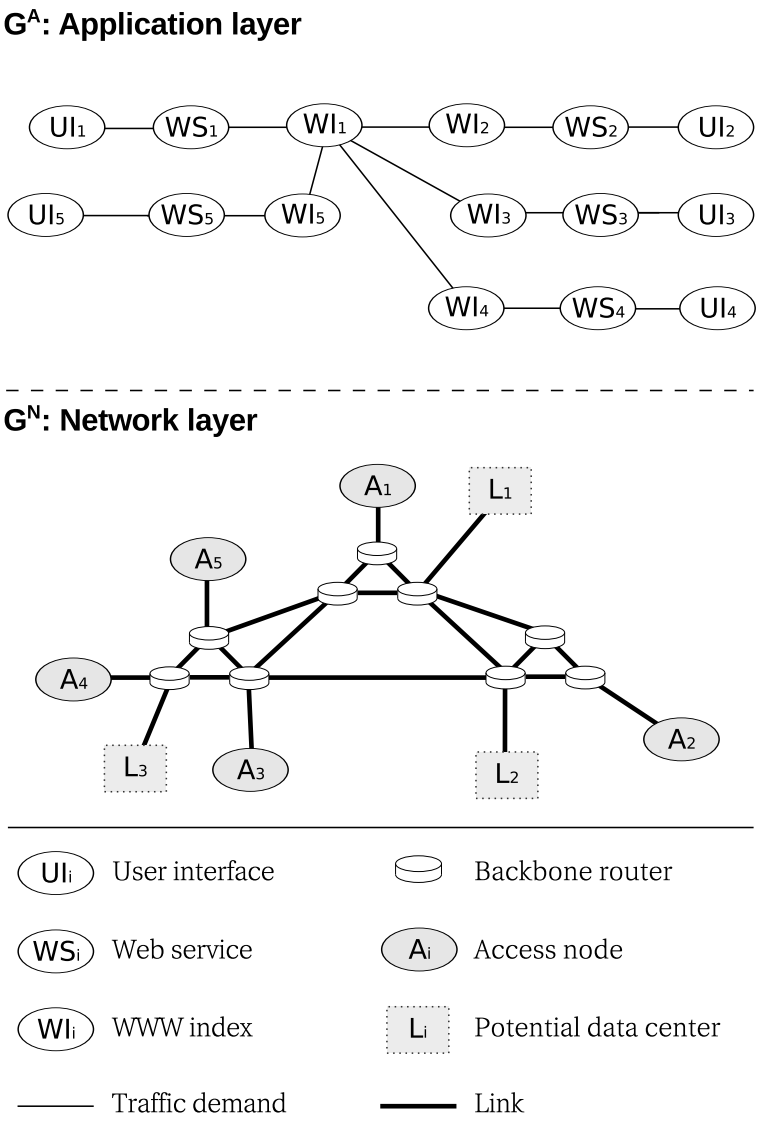

Figure 3.1 Network and application layers.

the quality of service is increased. That implies, for instance, that web service components will be placed near the users to reduce the round trip time. The following is the network traffic notation:

b_d^1 Throughput of demand $d \in A^A$ in peak time (in bps).

b^2 Capacity per link channel (in bps), e.g., 10 Gbps.

u_e Maximal utilization ratio per channel in link $e \in A^N$ (e.g., 0.70).

t_e Delay bound for link $e \in A^N$ (in seconds per packet) while its traffic is less than the allowed utilization.

3.3.2 Servers

Cloud computing applications have software components that are executed either on the user devices or on servers. In fact, a single software component may be executed on multiple servers. For instance, in a web search engine, multiple web servers answer HTTP requests. Each server has capacity allocated for each one of its resources, including network bandwidth, CPU, RAM, and hard disks. In this model, we consider that the number of replicas of each software component, i.e., the number of required servers, is a parameter that is estimated in advance and strictly depends on the nature of the application. The server-related notation is as follows:

K Set of resources that each server has, which include 1 = CPU, 2 = network card, 3 = RAM, and 4 = storage.

r_{ip}^1 Average consumption of resource $p \in K$ required by a copy of component $i \in V^A$, in the correspondent units, e.g., GHz, Gbps, or GB.

r_{ip}^2 Peak consumption of resource $p \in K$ required by a copy of component $i \in V^A$.

r_i^3 Number of replicas of software component $i \in V^A$.

r_p^4 Capacity of resource $p \in K$ on a server.

r_k^5 Capacity of data center $k \in \overline{\mathcal{D}}$ in number of servers.

r^6 Number of servers per Local Area Network (LAN) switch.

The servers and LAN switches in a data center are organized in racks, and each data center has space for a maximal number of racks. Each switch can connect a maximal number of servers, which will determine the number of required switches. In summary, each software component requires a specific number of servers. The model solution defines what software components each data center hosts, hence the number of servers required in each data center. From that, the number of LAN switches needed for each data center in a particular solution can be calculated. The sum of servers and switches must be less than the data center capacity.

3.3.3 Energy

An important objective of cloud computing is to minimize the power consumption of the entire network. The power consumption is primarily from the servers, the network switches, the routers, the data center cooling, and the power provisioning equipment. The server power consumption depends on its utilization, having a minimal idle power and reaching a maximal power consumption when the resources are fully utilized [72]. The following is the energy-related notation:

w^1 The power consumption of a server as a function of the utilization of each resource (in watts).

w^2 Network switch power consumption (in watts).

w^3 Router port power consumption (in watts).

w_k^4 Maximum power capacity of data center $k \in \overline{\mathcal{D}}$ (in watts) for IT equipment.

w_k^5 Power Usage Effectiveness (PUE) of data center $k \in \overline{\mathcal{D}}$ (ratio of total power / IT equipment power).

w_k^6 CO_2 emissions in data center $k \in \overline{\mathcal{D}}$ (in g / KWh).

w_i^7 Average power consumption of a server with a replica of software component $i \in V^A$.

w_i^8 Peak power consumption of a server with a replica of software component $i \in V^A$.

CO_2 emissions depend on how the energy is produced because each type of energy generates a certain amount of CO_2 per KWh. For example, hydroelectric energy emits 10 g of CO_2/KWh, and diesel emits 778 g/KWh [11]. Because the power consumption, w_1, depends on each piece of hardware of the server used, the values w_i^7 and w_i^8 should be determined experimentally.

3.3.4 Cost

The following are the parameters that define the cost of each cloud element, the cost of the energy and the penalties. Each element has a different amortization period; data centers are amortized between 12 and 15 years, and servers are amortized between 3 and 4 years [9]. Each cost below is the result of the total element cost divided by its amortization years.

c_d^1 Penalty for delay in demand $d \in A^A$
(in \$ / (ms/packet) / year).

c^2 Cost of a LAN switch (in \$/year).

c^3 Cost of a router port (in \$/year).

c_e^4 Cost of a channel in link $e \in A^N$ (in \$/year).

c^5 Cost of a server (in \$/year).

c_k^6 CAPEX for using data center $k \in \overline{\mathcal{D}}$ (in \$/year).

c_k^7 OPEX for using data center $k \in \overline{\mathcal{D}}$ (in \$/year).

c_k^8 Electricity cost in data center $k \in \overline{\mathcal{D}}$ (in \$/MWh).

c_k^9 Penalty for emitting CO_2 in data center $k \in \overline{\mathcal{D}}$ (in \$/ton).

A penalty is applied to the delay for exchanging information between the software components. Some traffic demands may suffer from the delay more than others. For example, a traffic demand could be used for copying files from a data center to calculate statistics once every day. That traffic demand might not be affected by an additional delay; therefore, its delay penalty is low. Traffic demands from the users doing search requests require a very short delay, and in that case, the penalty is high; this penalty represents a loss in revenue because users may change to the competitors.

The pollution penalty term comes from two sources. One is the country regulations that apply a carbon tax to the energy consumption. In that case, the electricity cost c_k^8 is set as the base cost without that tax in data center k, and c_k^9 includes the carbon tax. There is a second source of CO_2 penalty that may be even higher: the losses in corporate image. Nowadays customers reward companies that make efforts to reduce pollution and its quantification should be included in the CO_2 penalty.

3.3.5 Decision variables

x_{ik} 1 if software component $i \in V^A$ is located in a network node $k \in V^N$; 0 otherwise.

y_k 1 if node $k \in V^N$ hosts one or more software components; 0 otherwise.

z_k Number of servers hosted in data center $k \in \overline{\mathcal{D}}$.

s_k Number of LAN switches in data center $k \in \overline{\mathcal{D}}$.

f_{de} Amount of demand $d \in A^A$ traffic carried by link $e \in A^N$ (in bps).

b_e Number of channels, e.g., wavelengths or fibers, in link $e \in A^N$.

w_k Average power consumption of IT equipment in node $k \in V^N$.

The problem is modeled as a multi-commodity flow problem, in which each flow variable contains the amount of traffic per demand, per link. These variables are subsequently related to the location variables of the software components because moving a software component moves its traffic demands.

3.3.6 Objective function

Each criterion to optimize is included in a term of the objective function. The following are the formulas to calculate each cost. Note that some of these formulas contain constants to change units, e.g., seconds to milliseconds, KWh to MWh, or Gbps to bps.

The flow variables that define how demands are routed are used to calculate each demand delay, that is the end-to-end delay incurred by packets between software components. For instance, the delay of the demand between a user interface and a web service will be included in this computation. Then, each demand delay is multiplied by its delay penalty, and the total delay penalty is:

$$c^D = \sum_{d \in A^A} c_d^1 1000 \sum_{e \in A^N} t_e \frac{f_{de}}{b_d^1}$$

The network traffic cost is composed of the costs of the LAN switches, the router ports and the links. Each pair of link channels, uplink and downlink, of a link is connected to two router ports, one in each end of the link. Therefore, the number of router ports required for a link is equal to the number of channels.

$$c^T = \sum_{k \in V^N} c^2 s_k + \sum_{e \in A^N} (c^3 + c_e^4) b_e$$

The following is the cost of all servers in the network:

$$c^S = \sum_{k \in V^N} c^5 z_k$$

The capital expenditures (CAPEX) of each data center includes the land cost, the cost of building the facility, the cooling artifacts, and the power provisioning instruments. The total CAPEX of a solution is calculated as follows:

$$c^C = \sum_{k \in V^N} c_k^6 y_k$$

The data center operational expenditures (OPEX) includes the human labor, i.e., technicians and security guards, as well as the administrative costs. We exclude the electricity because we consider it in an explicit term.

$$c^O = \sum_{k \in V^N} c_k^7 y_k$$

The energy cost for a whole year is calculated as a function of the data center energy consumption, its PUE, and the price of energy as follows:

$$c^E = \sum_{k \in V^N} c_k^8 (w_k \cdot 365 \cdot 24\text{h} \cdot 10^{-6}) w_k^5$$

In a similar way, the environmental cost is proportional to the energy consumed in each data center, the ratio of CO_2 emissions, and the CO_2 emission penalty:

$$c^{CO_2} = \sum_{k \in V^N} c_k^9 (w_k^6 \cdot 10^{-6})(w_k \cdot 365 \cdot 24\text{h} \cdot 10^{-3}) w_k^5$$

Finally, each cost in the objective function is weighted by a parameter, allowing planners to modify the priorities. The objective function is defined as follows:

$$z = \alpha c^D + \beta c^T + \gamma c^S + \delta c^C + \epsilon c^O + \eta c^E + \psi c^{CO_2} \tag{3.1}$$

3.3.7 Constraints

• *Location of software components and data centers*

$$\sum_{(i,k)\in P} x_{ik} = 1 \qquad \forall \quad i \in V^A \tag{3.2}$$

$$\sum_{\substack{k\in V^N / \\ (i,k)\notin P}} x_{ik} = 0 \qquad \forall \quad i \in V^A \tag{3.3}$$

$$x_{ik} \leq y_k \qquad \forall \quad (i,k) \in P \tag{3.4}$$

Equations (3.2) and (3.3) state that each software component must be placed in one node, either access node, or data center. They also state that each assignment of a software component to a node must belong to the set of possible assignments P. Constraint (3.4) specifies that nodes containing software components are open.

• *Flow conservation*

$$\sum_{e\in\Gamma_k^-} f_{de} + x_{ik} b_d^1 = \sum_{e\in\Gamma_k^+} f_{de} + x_{jk} b_d^1 \tag{3.5}$$
$$\forall \quad d = (i,j) \in A^A, \quad k \in V^N$$

This constraint relates the software component location to the routing. It associates the application and the network layers. For each demand from software component $i \in V^A$ to software component $j \in V^A$ located in nodes $k_i, k_j \in V^N$, respectively, the traffic b_d^1 enters the network in k_i and exits in k_j. In each intermediate node k from k_i to k_j, flow is conserved.

• *Link capacity*

$$\sum_{d\in A^A} f_{de} \leq u_e \, b^2 \, b_e \qquad \forall \quad e \in A^N \tag{3.6}$$

$$b_{ij} = b_{ji} \qquad \forall \quad (i,j) \in A^N \tag{3.7}$$

Constraint (3.6) defines the link capacity variable b_e (in number of channels), requiring that the traffic is less than the maximal utilization threshold, $u_e \, b^2 \, b_e$. Because full-duplex links are assumed, equation (3.7) states that the capacity in both directions of the link is the same.

- *Number of servers*

$$z_k = \sum_{i \in V^A} r_i^3 x_{ik} \qquad \forall \quad k \in \overline{\mathcal{D}} \tag{3.8}$$

The number of servers required in each data center is the sum of the servers required by the software components that are located there.

- *Number of switches*

$$r^6 s_k \geq z_k \qquad \forall \quad k \in \overline{\mathcal{D}} \tag{3.9}$$

Because each LAN switch can connect a maximum number of servers r^6, Constraint (3.9) defines the minimal number of switches, s_k, required to connect the z_k servers in data center k.

- *Data center capacity*

$$z_k + s_k \leq r_k^5 \qquad \forall \quad k \in \overline{\mathcal{D}} \tag{3.10}$$

The number of servers and LAN switches must be less than the maximal capacity of the data center.

- *IT equipment average power consumption*

$$w_k = w^2 s_k + \sum_{e \in \Gamma_k^-} w^3 b_e + \sum_{i \in V^A} w_i^7 r_i^3 x_{ik} \; \forall k \in V^N \tag{3.11}$$

Equation (3.11) sums the average amount of power consumed by the server switches, the router ports, and the servers in a node. This value is used to calculate the cost of electricity used in the solution.

- *Data center maximal peak power*

$$w^2 s_k + \sum_{e \in \Gamma_k^-} w^3 b_e + \sum_{i \in V^A} w_i^8 r_i^3 x_{ik} \leq w_k^4 \quad \forall k \in \overline{\mathcal{D}} \tag{3.12}$$

The peak power consumption of data center k is calculated using the same methodology as for the previous constraint. This value has a fixed limit in each data center because of the electricity available and the power provisioning equipment in the data center.

- *Domain of variables*

$$x_{ik} \in \{0,1\} \quad \forall \quad i \in V^A, k \in V^N \tag{3.13}$$

$$y_k \in \{0,1\} \quad \forall \quad k \in V^N \tag{3.14}$$

$$z_k \in \mathbb{Z}_{\geq 0} \quad \forall \quad k \in \overline{\mathcal{D}} \tag{3.15}$$

$$s_k \in \mathbb{Z}_{\geq 0} \quad \forall \quad k \in \overline{\mathcal{D}} \tag{3.16}$$

$$b_e \in \mathbb{Z}_{\geq 0} \quad \forall \quad e \in A^N \tag{3.17}$$

$$f_{de} \in \mathbb{R}_{\geq 0} \quad \forall \quad d \in A^A, e \in A^N \tag{3.18}$$

$$w_k \in \mathbb{R}_{\geq 0} \quad \forall \quad k \in V^N \tag{3.19}$$

Each variable domain is defined. All the variables are non-negative. The location variables may take the values zero or one. The number of servers, LAN switches, and network channels are integers. The flow and the data center power consumption variables are real numbers.

3.4 Solution approach

This section describes the tabu search heuristic [13, 73] proposed to solve the cloud network planning problem. The algorithm was programmed in C++ and compiled using gcc 4.5.1 with the optimization options activated. The quality of solutions obtained and the execution time are compared with the MILP model in AMPL with Cplex 12.4. Algorithm 1 presents the main structure of the proposed tabu search heuristic.

3.4.1 Solution space

A solution of the proposed tabu search heuristic is a mapping of the software components to the network nodes, i.e., M: $V^A \rightarrow V^N$. In that solution M, each software component is placed at one of the network nodes. P specifies the set of possible assignments; hence, for each software component i, the pair $(i, M(i))$ must belong to P. Thus, the solution space of the proposed heuristic is the set of mappings, as follows:

$$\mathcal{S} = \left\{ M : V^A \rightarrow V^N / \, (i, M(i)) \in P, \quad \forall i \in V^A \right\}$$

Each solution M determines the location variables x_{ij} and satisfies the constraints (3.2) and (3.3) in the model. From the software component placement defined by variables x_{ij}, all the other variables of data center location (y_j), data center capacities (s_k and z_k), routing (f_{de}), link capacities (b_e), and power consumption (w_k) can be calculated in a deterministic way as follows. The variables y_j of the data centers hosting the software components have a

Algorithm 1 Tabu search for the cloud network planning problem.

function TabuSearch
  $M \leftarrow$ InitialGreedySolution
  $Best \leftarrow M$
  $L \leftarrow \{\}$
  $i \leftarrow 0$
  repeat
    Choose $(c, d) \in P - L$ that minimizes z when moving software component c to node d in M.
    Move c to d in M
    Add (c, d) to tabu list L for $\frac{\sqrt{|\mathcal{N}(\mathcal{M})|}}{2}$ iterations.
    $i \leftarrow i + 1$
    if $z(M) < z(Best)$ **then**
      $Best \leftarrow M$
      $i \leftarrow 0$
    end if
  until $i = MAX\_IT$
  return $Best$
end function

function InitialGreedySolution
  $M \leftarrow \emptyset$
  for each component c in V^A **do**
    $d \leftarrow x/(c, x) \in P$ and x is the node that least increases z when placing c in x in M.
    $M \leftarrow M \cup \{(c, d)\}$
  end for
  return M
end function

value of 1, as stated by constraint (3.4). Because the software component location is defined in M, the origins and destinations of the traffic demands are fixed. We use the shortest path between each pair of nodes of V^N for routing the demands and defining variables f_{de} that satisfy the flow conservation constraint (3.5). The distance considered in the shortest path algorithm is the delay bound t_e of each link. The routing defines the total amount of traffic carried by each link; therefore, we can calculate the minimal number of channels required for that link using the channel capacity, c_e, and the maximal allowed utilization ratio per channel, u_e, satisfying constraints (3.6) and (3.7). The numbers of servers, z_k, and switches, s_k, are also determined by the software components hosted in data center k through the constraints (3.8) and (3.9). Variables w_k that correspond to the amount of power consumed by the IT equipment in data center k are calculated with the formula defined in equation (3.11). Finally, the terms of the objective function that correspond to a delay of traffic demands, c^D, network traffic c^T, servers c^S, CAPEX c^C, OPEX c^O, electricity c^E, and CO_2 emissions c^{CO_2}, are determined, and the value of the objective function is defined in equation (3.1). Thus, we see that the location of the software components determines all the variables in the model, and subsequently all the terms of the objective function.

The solutions in $\mathcal{S}$ satisfy all the constraints except the data center capacity constraints (3.10) and (3.12). Instead of enforcing these constraints, one penalty term for each constraint is added to the objective function. The number of servers that exceeds the capacity in an overloaded data center is multiplied by a large constant, which is higher than all the other terms of the objective function. The same penalty is applied for power consumption that is higher than the data center limit. Thus, the tabu search will avoid these solutions because the objective function is much higher than in solutions that do not have overloaded data centers. The algorithm will thus accommodate corner cases, which have capacities that make feasible solutions difficult to find. In that case, the tabu search will search to reduce the overloaded data centers until a feasible solution is found.

3.4.2 Initial solution

The greedy heuristic shown in Algorithm 1 locates one software component of V^A at a time. Each step chooses the node in V^N where the objective function has the lowest value when the software component is placed in that node. Because the objective function includes delay, cost, energy, and CO_2 the node chosen will be the best in these aspects, according to the priorities defined. If the first priority is the delay, a web service will be hosted in the closest data center to its users. If it is the CO_2, then the data center with the cleanest energy will be chosen. As in every greedy heuristic, the placement is done in an arbitrary order, and

that may be non optimal. The greedy algorithm provides a good starting solution that the tabu search improves to a near optimal solution.

Figure 3.2 shows a greedy solution for the example of a web search engine. In the first five steps, each one of the user interfaces UI_i is located in its correspondent node A_i, defined in the set P. In those steps, there is no choice because each UI_i has a unique possible assignment in P. Until that time, the objective function remains zero because those components do not require servers and do not have direct traffic demands among them. Then, the location of the web service WS_1 that serves the requests from UI_1 is the node that increments the objective function the least. If all the data centers have the same costs and capacities and all the links have the same delay bound, the best possible data center for WS_1 is L_1, as shown in Figure 3.2. That is because UI_1 is located in A_1 and there is a traffic demand between UI_1 and WS_1 that contributes a delay that is part of the objective function. The shortest path between A_1 and L_1 has three links, and the shortest paths between A_1 and the other data center locations L_2 and L_3 have more links; hence more delay, given that all the links have the same delay. In the same way, WS_2, WS_3, WS_4 and WS_5 are located in the data center locations that are closest to the respective user interface. Given that the web indexes WI_i exchange a high amount of traffic with the corresponding web services WS_i, the locations that have a smaller increment in the objective function are those where the web index is in the same data center as the web service.

3.4.3 Neighborhood relation

Each iteration of the tabu search moves from the current solution to a neighbor solution. One solution M is neighbor of a solution M' if and only if they differ in the location of only one software component i. The neighborhood of solution M is the following set:

$$\mathcal{N}(M) = \left\{ M' \in \mathcal{S} / \exists!\ i \in V^A, M'(i) \neq M(i) \right\}$$

Thus, one software component is moved from one node to another in each iteration of the tabu search. The chosen movement is that with the lowest objective function. The calculation of the objective function for each neighbor can be time consuming for large instances because the number of possible movements is high. However, the key of the tabu search heuristic is that each iteration must be performed as rapidly as possible. Calculating the objective function in an incremental way achieves that efficiency. Instead of performing the entire calculation for each neighbor, the difference between the current solution and the neighbor solution is calculated. That is, the amount that the objective function would change if a component is moved is what is calculated. In other words, the difference of each objective

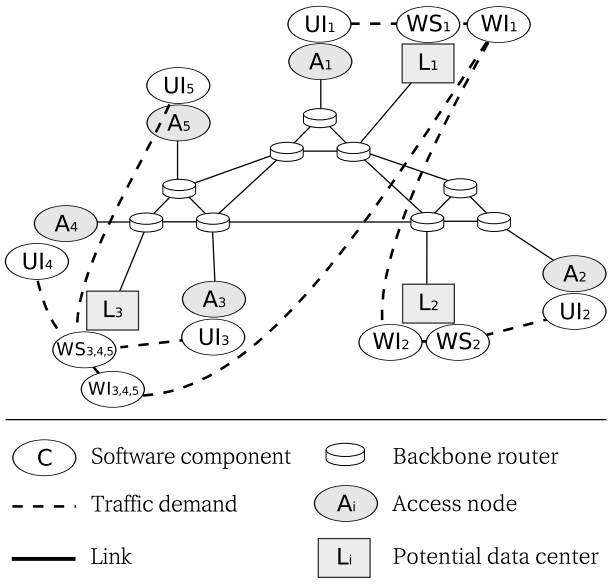

Figure 3.2 Greedy solution for the web search engine example.

function term is calculated. Coding this approach with data structures with constant access time is more complicated than the straightforward calculation of the objective function, but the execution time of each iteration is several orders of magnitude shorter when performed incrementally. That means that the tabu search heuristic moves very rapidly among the solutions.

Limiting the number of neighbors to analyze can also reduce the execution time of each iteration. Instead of looking at the entire neighborhood, the tabu search analyzes a random subset of them [74]. The number of neighbors to analyze is $4\sqrt{|N(M)|}$, a number that grows as the instance size increases but is less than a linear function. The multiplier 4 was adjusted through experiments, making a trade-off between the execution time and the solution quality obtained in each iteration.

3.4.4 Tabu list

The risk of this type of heuristic is to reach a local optimum and keep returning to it iteration after iteration. A mechanism to avoid repeated solutions prevents that situation. When a software component i is moved to a node j, then the pair (i, j) is added to a tabu list, L, so that movement will not be performed again for a number of iterations. This mechanism effectively avoids local optimums and the searching process continues. The number of iterations to keep the movement in L is related to the neighborhood size and is equal to $\frac{\sqrt{|\mathcal{N}(\mathcal{M})|}}{2}$. This value was carefully adjusted to grow when the neighborhood is larger, but in a less degree than a linear function.

3.4.5 Stop criterion

If the best solution found does not improve after *MAX_IT* iterations, then the algorithm stops. *MAX_IT* is defined as the sum of the number of nodes in the network and the number of software components, i.e., $|V^N| + |V^A|$.

3.5 Case study

In this section, we study the deployment of a hypothetical web search engine on a network of data centers.

3.5.1 Network and application topology

The network nodes V^N include a set of access nodes $\mathcal{A}$, backbone routers $\mathcal{B}$, and locations for new data centers $\mathcal{L}$. The nodes are distributed among different locations around the globe. To do that, we considered the 500 largest cities in the world [75], the number of Internet users in each of them [76], and their geographic location [77]. For each city, we created an access node in $\mathcal{A} \in V^N$ that aggregates all the traffic of its users. The access nodes are connected through intermediate backbone routers. Finally, for each one of these nodes—routers and access nodes—, a potential data center location was created in $\mathcal{L}$. This method allows us to produce scenarios of different sizes, with up to 500 access nodes and 1000 potential data center locations.

Regarding the application layer G^A, we used the web search engine topology shown in Figure 3.1 for n access nodes. There is a user interface UI_i for each access node that aggregates every web client in that city. There is also a web service WS_i that answer the users' HTTP requests and make queries to the web indexes WI_i. All the web indexes keep synchronized with a master web index. The decisions to make are which data centers will host each web service and web index, as well as all the network planning variables of the proposed model, to minimize the delay, the cost, energy consumption, and the CO_2 emissions.

3.5.2 Network traffic

The total inter-domain traffic of the Internet was 154 Tbps in March 2013, considering the estimation of 39 Tbps in July 2009 [78] and a 45.5% annual growth. The hypothetical web search engine analyzed manages 1% of the Internet traffic, that is 1.54 Tbps. That total amount of traffic was distributed among the access nodes in proportion to the number of Internet users in each city [76]. These are the traffic demands between the user interfaces UI_i and the web services WS_i (b_d^l). The same value was used between each WS_i and its WI_i.

The average packet size is 770 bytes (ζ), the capacity of each link channel is 10 Gbps (b^2), the maximal channel utilization is 70% (u_e), and the maximal link delay is 1 ms plus the propagation delay (t_e). The propagation delay of each link is calculated with the distance between the nodes divided by the speed of light over optical fiber cable (200,000 km/s).

3.5.3 Servers

The number of servers required is 200,000, also distributed in proportion to the number of users in each city. From the servers required by each city, 60% are for web services and 40% for the web index (r_i^3). Each data center can accommodate either 40, 80, 160, 1,000, 32,000, 64,000, or 128,000 IT elements. Each LAN switch can connect up to 40 servers (r_{ip}^6).

3.5.4 Power

The average power consumption of each server (w_i^7) was defined as 150 W, and the peak power consumption (w_i^8), as 300 W. Each LAN switch consumes 400 W (w^2), and each 10 Gbps router port consumes 20 W (w^3). The data center power capacity for IT equipment (w_k^4) is either 12 KW, 24 KW, 49 KW, 303 KW, 9.7 MW, 19.4 MW, or 38.7 MW, depending on the data center size. The PUE of the data centers is 1.08 (w_k^5), that is the power provisioning equipment, cooling artifacts and any other power consumption are 8% of the power consumed by the IT equipment. The type of energy that supplies a data center generates a specific amount of CO_2 emissions (w_k^6): offshore wind produces 9 g / KWh, hydroelectric 10 g / KWh, geothermal 38 g / KWh, nuclear 66 g / KWh, natural gas 443 g / KWh, diesel 778 g / KWh, and coal 960 g / KWh [11]. The amount of CO_2 produced by each data center in this case was randomly chosen from one of these values.

3.5.5 Cost

The delay penalty was defined as \$10,000 / (ms / packet) / year for the traffic demands between the UI_i, WS_i, and WI_i, and \$10 / (ms / packet) / year for the delay between the WI_i themselves (c_d^1). That means that the delay between UI_i, WS_i, and WI_i is much more important than the delay between the WI_i themselves. The cost of each LAN switch is \$1,000 / year (c^2). Each 10 Gbps router port costs \$100 / year (c^3) and each directed 10 Gbps link channel, \$60,000 / year (c_e^4) [79]. The cost of each server is \$4,000 [80] and it is amortized in 4 years, resulting in \$1,000 / year (c^5). The data center CAPEX is \$12 / W amortized in 12 years giving a total of \$1 / W / year (c_k^6), and the OPEX is \$0.24 / W / year (c_k^7) excluding electricity that was considered as an explicit term of the cost function [9].

Electricity prices were uniformly distributed between \$30 / MWh and \$70 / MWh (c_k^8) [81]. Finally, the penalty for emitting CO_2 was defined as \$1,000 / ton (c_k^9).

3.6 Results

3.6.1 Trade-off between objectives

As a first step, we executed the AMPL-Cplex model on a case of 10 cities to analyze how the different aspects of the model interact. This case will also demonstrate how planners can use the framework with multiple objectives. The network has 10 access nodes and 10 potential data center locations. In Figure 3.3, the access nodes are shown as black circles and the potential data center locations are shown as white boxes. The application graph is a web search engine with a user interface, a web service, and a web index for each access node, and the topology of traffic demands as discussed in Section 3.5.

The case was solved in three different contexts, depending on the optimization priorities: A) delay oriented, B) pollution oriented, and C) cost oriented. Table 3.2 presents the results of the three solutions, and Figures 3.4, 3.5, and 3.6 show the data centers selected in each context. All the tests were executed on an Intel i7 with 4 cores at 2.8 GHz.

- In Solution A, 10 data centers are open, i.e., all of them are used to host servers and software components. That optimization initially favors the delay because the delay penalty is the highest term in the objective function at \$ 856.2 M. In this case, all the access nodes have a close data center that serves their requests, and the average delay is 4.4 ms. The total power consumed by all the data centers is 34.6 MW. The data centers are powered with all types of energy, in particular, 4 of them have high pollution emission ratios, between 443 and 960 grams of CO_2 per KWh. The total CO_2 emission is 74,695 ton per year, and the total cost is \$ 678.1 M per year.

Figure 3.3 Network topology with 10 cities and 10 potential data centers.

Figure 3.4 Solution A. Delay minimization.

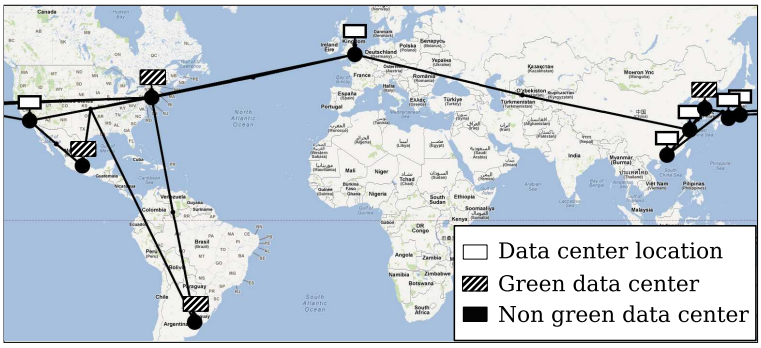

Figure 3.5 Solution B. Pollution minimization.

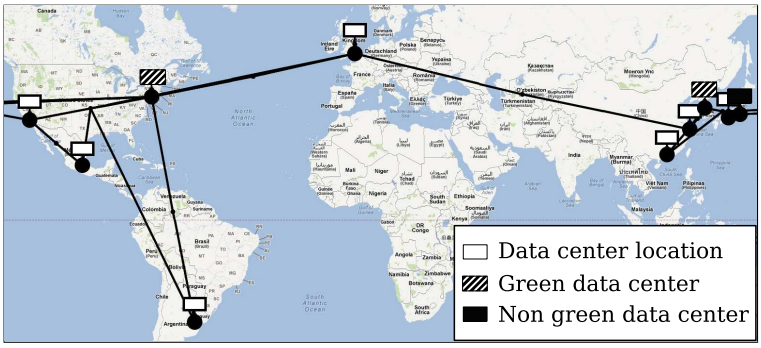

Figure 3.6 Solution C. Cost minimization.

Table 3.2 Solutions for 10 cities with up to 10 data centers.

Solution:	A	B	C
Delay multiplier	1	1	1
CO_2 multiplier	1	1000	1
Cost multiplier	1	1	1000
Delay	**4.4 ms**	12.2 ms	21.2 ms
Total power	34.6 MW	34.6 MW	34.6 MW
CO_2 emissions	74,695 ton	**3,561 ton**	41,181 ton
Data centers	10	4	3
Delay penalty	**856.2 M**	2,563.5 M	5,336.2 M
CO_2 penalty	74.7 M	**3.6 M**	41.2 M
Traffic cost	52.4 M	78.0 M	77.0 M
Server cost	200.0 M	200.0 M	200.0 M
CAPEX	329.1 M	135.5 M	77.4 M
OPEX	79.0 M	32.5 M	18.6 M
Energy cost	17.7 M	16.8 M	17.5 M
Total cost	678.1 M	462.8 M	**390.5 M**
Execution time	0.7 s	0.4 s	178 s

- Solution B is obtained by augmenting the pollution multiplier in the objective function (ψ) to 1000. That makes the CO_2 penalty term greater than the delay penalty term. In this case, the optimal solution is to open 4 data centers. Three of the data centers are powered with hydro-electric energy producing 10 grams of CO_2 per KWh, and one of the centers is powered with geothermal energy introducing 38 grams of CO_2 per KWh. The total CO_2 emissions are 3,561 ton per year, 21-times smaller than Solution A. The CO_2 penalty decreases with the same ratio. The energy consumption remains the same because most of it depends on the number of servers used and the PUE of the data centers, and in these cases, both remain constant. The total cost is 68% of Solution A because fewer data centers are used; therefore, the CAPEX and the OPEX are reduced. The average delay is 12.2 ms, which is 2.8-times greater than Solution A.

- Solution C is obtained by setting the cost multipliers to 1000, making the total cost the first priority to optimize. The number of open data centers is 3, and they have the minimal capacity to accommodate all the servers. The total cost is reduced to $ 390.5 M, the total CO_2 emissions are 41,181 tons per year, and the average delay is 21.2 ms.

From the three solutions, Solution B appears to be a good compromise between the network performance, pollution, and cost because the four chosen data centers imply a short delay with an acceptable cost and very low CO_2 emissions. Using this framework, planners

can evaluate multiple alternative solutions and analyze the trade-offs between them. Furthermore, once the optimal values for each aspect are known, additional constraints can be added to evaluate solutions around the optimal values. The execution time to find these solutions was between 0.7 s and 178 s. We will see how the Cplex execution time increases very quickly as the network growths and how the tabu search performs very well for small and large cases.

3.6.2 Comparison between tabu search and Cplex

We executed the tabu search heuristic and the AMPL-Cplex model in eleven small cases to evaluate the execution time and the solution gaps in the tabu search with respect to the optimal values. The small cases had from 2 to 12 access nodes and from 4 to 24 potential data center locations. In these cases, the data center capacities were set at either 64,000 or 128,000 to be able to host 200,000 servers at a small number of data centers. Table 3.3 shows the results when the optimization priority is the delay, Table 3.4 shows the results with pollution minimization, and Table 3.5 presents the results when the priority was the network cost. All the costs are expressed in millions of dollars. Each case was executed 10 times using the tabu search heuristic to calculate the averages of each value, as well as the minimal, the average, and the maximal gaps. The minimal gaps are between 0 and 1.95%; most of them are below 0.1%, showing that the heuristic is very effective in finding near-optimal solutions. The last column of the tables exhibits the gap between the value found with the greedy heuristic and the optimal value. In two cases, greedy found the optimal values but most of the cases were above 10%. That reveals that the tabu search is what effectively improves the solution to near optimality.

The execution time of the tabu search ranged from less than 1 millisecond to 868 milliseconds, whereas Cplex used more than 58 minutes for the most difficult case. For that complex case, with 12 access nodes, 24 data centers and cost minimization, the tabu search heuristic had an average gap of 1.95% and an average execution time of 868 milliseconds. The results also show that the cost-oriented optimization requires more execution time in all the cases. This extra execution time is needed because there are many near-optimal solutions that cannot be discarded until the optimum is found. In the delay-oriented case, the feasible solutions with small delay are those with servers near the users. Once one of these solutions is found, the components will tend to be fixed near the users, and other solutions are discarded, reducing the search space and the execution time. In the case of CO_2 optimization, feasible solutions with low emissions will discard data centers with high emissions. It is not possible to discard solutions in the cost oriented case because many data centers have the same CAPEX and OPEX, resulting in many combinations with similar costs.

Table 3.3 Comparison between tabu search and Cplex. Delay priority.

#AN	#DC	Tabu(s)	Cplex(s)	Delay(ms)	CO_2(ton)	Cost	z	Min.gap	Avg.gap	Max.gap	GR.gap
2	4	0.001	0.032	2.9	6,520	360.0	593.8	0.00%	0.00%	0.00%	0.00%
3	6	0.002	0.090	2.6	49,473	385.7	748.6	0.00%	0.00%	0.00%	24.44%
4	8	0.013	0.287	2.8	101,757	402.9	926.3	0.00%	0.00%	0.00%	22.44%
5	10	0.023	0.137	2.4	90,729	462.9	1,060.4	0.00%	0.00%	0.00%	0.38%
6	12	0.061	0.272	2.2	43,688	484.0	1,059.9	0.02%	0.02%	0.02%	0.02%
7	14	0.071	0.381	2.3	131,135	509.9	1,279.1	0.02%	0.02%	0.02%	1.64%
8	16	0.165	0.532	2.5	130,208	583.0	1,477.1	0.02%	0.08%	0.62%	0.62%
9	18	0.158	0.564	2.2	96,281	629.1	1,495.3	0.02%	0.02%	0.02%	0.02%
10	20	0.276	0.719	2.3	112,688	606.8	1,614.1	0.01%	0.01%	0.01%	0.01%
11	22	0.313	0.822	2.5	143,455	626.0	1,836.2	0.01%	0.01%	0.01%	0.01%
12	24	0.373	0.996	2.5	36,043	675.5	1,820.1	0.01%	0.01%	0.01%	0.01%

Table 3.4 Comparison between tabu search and Cplex. CO_2 priority.

#AN	#DC	Tabu(s)	Cplex(s)	Delay(ms)	CO_2(ton)	Cost	z	Min.gap	Avg.gap	Max.gap	GR.gap
2	4	0.002	0.031	2.9	6,520	360.0	7,138.7	0.00%	0.00%	0.00%	4.60%
3	6	0.005	0.130	13.5	5,645	408.8	8,085.0	0.00%	0.00%	0.00%	9.41%
4	8	0.022	0.361	8.6	18,178	393.8	19,983.1	0.00%	1.07%	10.22%	13.14%
5	10	0.033	0.336	6.6	6,108	435.4	7,998.7	0.00%	0.00%	0.00%	22.28%
6	12	0.047	0.354	10.3	4,650	393.4	8,042.6	0.00%	0.96%	1.59%	1.59%
7	14	0.137	1.084	22.9	6,932	416.6	13,509.8	0.00%	1.67%	4.40%	29.38%
8	16	0.247	0.846	7.4	3,427	429.7	6,437.7	0.00%	0.00%	0.00%	9.47%
9	18	0.230	0.580	16.7	5,151	501.5	10,339.3	0.00%	0.59%	4.86%	26.63%
10	20	0.458	4.146	11.9	4,040	432.5	8,714.3	0.00%	0.23%	0.95%	15.39%
11	22	0.303	3.297	7.9	4,149	435.8	8,480.7	0.03%	0.03%	0.03%	0.03%
12	24	0.559	0.763	4.6	3,817	512.2	6,456.5	0.00%	0.00%	0.00%	0.00%

Table 3.5 Comparison between tabu search and Cplex. Cost priority.

#AN	#DC	Tabu(s)	Cplex(s)	Delay(ms)	CO_2(ton)	Cost	z	Min.gap	Avg.gap	Max.gap	GR.gap
2	4	0.001	0.045	2.1	170,026	357.6	357,907.8	0.00%	0.00%	0.00%	16.09%
3	6	0.005	0.283	9.9	87,891	362.9	364,469.4	0.00%	0.45%	4.47%	20.02%
4	8	0.010	1.163	6.8	210,694	364.9	366,263.7	0.00%	0.28%	0.94%	24.85%
5	10	0.033	2.422	10.8	143,705	374.9	377,336.6	0.00%	1.90%	3.56%	22.88%
6	12	0.076	5.361	9.6	60,234	365.3	368,031.9	0.10%	0.29%	1.75%	24.28%
7	14	0.124	58.147	9.9	184,332	365.3	368,744.8	0.10%	0.75%	3.19%	19.42%
8	16	0.205	83.514	10.0	138,276	373.8	377,319.7	0.06%	1.99%	6.38%	36.51%
9	18	0.288	40.673	8.7	50,595	367.0	370,745.7	0.03%	2.45%	4.45%	37.35%
10	20	0.460	1057.181	11.8	75,585	369.8	374,711.4	0.03%	2.00%	5.71%	28.61%
11	22	0.867	1496.597	12.1	141,955	375.2	381,295.8	1.05%	3.27%	7.02%	29.95%
12	24	0.868	3482.393	13.5	59,401	380.5	387,864.8	1.95%	4.53%	5.74%	30.69%

Table 3.6 Tabu search results for large cases. Delay priority.

#AN	#DC	Exec.(s)	#DC open	Iterations	Delay(ms)	CO_2(ton)	Total cost	z	GR.gap
50	100	0.4	34	405	3.9	91,330	1,188.3	8,762.1	0.00%
100	200	2.5	73	908	3.6	106,711	1,833.2	14,594.4	7.06%
150	300	5.7	117	1,212	3.9	108,790	2,611.4	21,098.7	0.85%
200	400	12.1	137	1,669	4.4	99,655	2,976.8	27,455.9	3.81%
250	500	20.8	199	2,149	4.7	79,202	3,735.3	32,103.8	3.35%
300	600	32.3	218	2,631	5.0	97,388	3,597.0	38,335.7	5.03%
350	700	45.6	261	3,061	5.3	103,037	4,201.3	43,899.6	4.67%
400	800	66.1	306	3,681	5.7	101,242	4,859.2	49,963.2	10.34%
450	900	85.4	356	4,047	6.2	92,121	5,078.5	54,795.5	10.05%
500	1000	117.5	396	4,718	6.6	116,096	5,349.3	60,327.2	12.41%

Table 3.7 Tabu search results for large cases. Pollution priority.

#AN	#DC	Exec.(s)	#DC open	Iterations	Delay(ms)	CO_2(ton)	Total cost	z	GR.gap
50	100	0.7	20	623	5.9	5,714	874.8	18,156.8	8.44%
100	200	2.2	52	806	4.8	5,344	1,259.0	23,013.4	0.74%
150	300	6.2	69	1302	5.0	6,982	1,848.2	33,031.9	6.37%
200	400	12.7	93	1744	5.1	7,500	2,330.5	40,687.0	4.05%
250	500	18.9	138	2027	5.3	5,746	2,765.8	42,423.4	0.67%
300	600	31.1	164	2658	5.5	7,171	3,183.9	50,408.5	3.40%
350	700	46.5	189	3094	6.0	7,620	3,705.2	58,609.4	3.41%
400	800	67.0	243	3712	6.4	8,609	4,267.8	65,433.0	7.98%
450	900	86.2	281	4182	6.8	8,325	4,737.6	69,195.0	12.77%
500	1000	112.4	314	4582	7.6	9,944	5,032.5	77,003.9	7.59%

Table 3.8 Tabu search results for large cases. Cost priority.

#AN	#DC	Exec.(s)	#DC open	Iterations	Delay(ms)	CO_2(ton)	Total cost	z	GR.gap
50	100	1.4	7	1,295	17.6	101,729	426.2	463,528.1	33.84%
100	200	9.6	29	3,342	24.8	101,188	466.2	544,064.0	46.72%
150	300	19.7	44	3,905	22.3	82,974	470.9	590,651.9	46.45%
200	400	52.1	52	6,762	23.4	96,500	494.2	662,006.5	61.72%
250	500	126.8	85	12,418	18.5	149,779	505.1	637,902.5	56.34%
300	600	190.3	122	15,330	21.1	123,173	528.7	695,767.9	58.15%
350	700	258.3	176	17,098	22.6	116,601	554.6	795,113.5	50.70%
400	800	459.0	208	24,552	21.6	59,005	584.5	781,621.2	29.18%
450	900	467.2	296	21,193	23.6	76,514	617.3	863,840.6	13.77%
500	1000	553.8	359	21,329	26.1	134,860	670.2	951,827.6	16.10%

3.6.3 Large cases

The tabu search heuristic is very efficient for small and medium problems. We will analyze larger cases, i.e., 50 to 500 access nodes and 100 to 1000 potential data center locations, to gauge the performance of the tabu search heuristic for large problems. Considering a large number of cities makes it possible to guarantee a good quality of service for most of the Internet users around the world. Large networks, such as Akamai, have small data centers in more than a thousand locations around the world to guarantee a high quality of service [12], which is why it is important to analyze the large cases using an efficient heuristic. In these cases, the data center sizes were 40, 80, 160, 1,000, 32,000, 64,000, or 128,000 IT elements; solutions may contain multiple small data centers or a combination of small, medium and large data centers. Tables 3.6, 3.7 and 3.8 show the network delay, costs, optimal values, and execution times for these cases. Each reported value is the average of 10 executions.

The results show that the execution time was less than 1 minute for the cases with up to 200 access nodes and 400 potential data centers, and it was less than 10 minutes in every instance. The delay was held to approximately 5 ms in the delay-oriented optimization by adding more data centers. The smallest case opened 34 data centers, and the largest case opened 396, i.e., 12-times more, but the cost was only multiplied by 4.5 because smaller data centers can be used. However, the CO_2 emissions were high compared to the pollution minimization cases. The average CO_2 in the first table is 99,557 tons, and it is 7,295 tons in the second table. The average delay in the CO_2 optimization only increased to 5.8 ms, which is less than 1 ms more than the delay-oriented optimization. The average cost decreased from $ 3,543.0 M to $ 3,000.5 M, which are both very large compared to the average optimal cost of $ 531.8 M in the third case. In that cost oriented strategy, the average cost was 18% of the cost of the CO_2 strategy, though the average delay increased to 22 ms, more than four-times that in the delay-oriented strategy.

In these cases, that are too large for Cplex to solve to optimality, we show the performance of the tabu search compared to the initial greedy solution. Here again the results exhibit a similar behavior to the near optimal results achieved in the small instances because the tabu effectively improves the initial solution: the gap between the greedy solution and the best tabu solution is up to 12.77% in Tables 3.6 and 3.7, and up to 61.72% in Table 3.8. To illustrate the gap, the cost oriented case AN=200, DC=400 had a total cost of $ 494.2 M in tabu search and $ 630.2 M in greedy. The difference can also be seen in Figure 3.7 where the values obtained by the tabu and greedy algorithms in the cost oriented cases are shown.

To our knowledge, these results exhibit the most flexible and efficient algorithm for data center location published in the literature. The flexibility is given by the possibility of defining an application graph that matches the forecasted traffic of the applications, and not only the

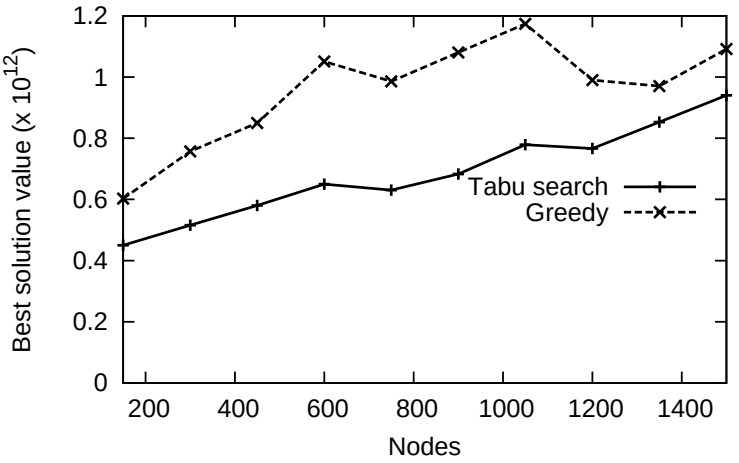

Figure 3.7 Tabu search vs greedy heuristic for large cases and cost priority.

traffic between users and servers. The model is also flexible in the sense of allowing planners to define the priorities in the multi-criteria objective function. The efficiency between different approaches is difficult to compare because each one solves a variant of the problem and for different cases. Nevertheless, we can point that a simulated annealing heuristic combined with linear programming was tested by Goiri et al. [14] with up to 500 data center locations and execution times of 30 minutes in a similar environment. Our case with 1000 data centers was solved by tabu search in 10 minutes. The solution quality cannot be compared because optimal solutions cannot be guaranteed in neither of the two approaches. We understand that our approach achieves such results because each tabu search iteration is extremely quick compared to expensive linear programming solutions. That was accomplished by using specialized data structures to calculate the objective function in an incremental way.

3.7 Conclusion

Cloud computing is expanding to increasingly more users, applications and devices; therefore, the network needs to grow in an efficient and sustainable way. In this paper, we presented a cloud network planning problem through a mixed integer-linear programming model. The location of data centers, servers, and software components impacts the network efficiency, the pollution, and the total cost. The definition of the network link capacities and the information routing is also very important. The proposed model allows planners to evaluate different solutions and to make variations in the optimization priorities.

The AMPL-Cplex implementation is useful for analyzing small cases, but its algorithmic complexity makes the execution time too high for real instances. The deployment of global networks needs the evaluation of the traffic from many cities around the world, and hundreds

of potential data center locations. We addressed that issue by developing an efficient tabu search heuristic. The tabu search algorithm evaluates potential neighbor solutions by calculating the difference from the current solution using the right data structures. The results showed that the tabu search heuristic could find near optimal solutions in a very short execution time. Instances of up to 500 access nodes and 1000 potential data center locations were solved in less than 10 minutes, showing that every real instance can be modeled and solved in this fashion.

The mathematical model and the algorithm can be extended in multiple directions. One direction is the server virtualization, a key aspect of cloud computing. With that feature, each server can host multiple independent virtual machines. Then, fewer servers are used, reducing costs and energy consumption. Another possible extension is to consider dynamic demands. In that case, the variations of the resource requirements in each hour of the day and on each day of the week are considered; therefore, the solutions make better use of the data centers, servers, and link capacities.

CHAPTER 4

ARTICLE 2: GREEN CLOUD BROKER, DYNAMIC VIRTUAL MACHINE PLACEMENT ACROSS MULTIPLE CLOUD PROVIDERS

Federico Larumbe and Brunilde Sansò

Submitted to
IEEE Transactions on Cloud Computing
on August 27, 2013

Abstract

Cloud computing aims to control the increasing energy consumption of applications by consolidating them in shared servers through virtualization. This technique can be greatly improved and complemented with a process called meta-scheduling that chooses the optimal data center for each Virtual Machine (VM). This paper proposes a dynamic optimization problem for the VM placement across multiple data centers to minimize operational expenditures, as well as improve quality of service (QoS) and reduce pollution. In fact, delay-sensitive applications with geographically distributed users experience varying response times depending on where users and VMs are located. Users closer to VMs experience less delay, thus distributing VMs close to users improves the QoS. On the other hand, the increasing energy consumption of the cloud raised concerns about the impact on CO_2 emissions and global warming. Choosing data centers that use green energy sources is an important way to mitigate this problem. These multiple aspects were embedded in a Mixed Integer Linear Programming (MILP) model that minimizes the cost of placing VMs across multiple data centers while respecting constraints in QoS, power consumption, and CO_2 emissions. The problem was solved through an efficient tabu search heuristic that can be used to deploy cloud applications with dynamic demands in real time. The optimization was executed for a network with more than 1,000 nodes, 650 applications, and 6,500 VMs. Results report that the communication delay is reduced up to 6 times when a meta-scheduler optimally chooses among multiple data centers instead of using the less expensive one. Furthermore, 30% of power consumption is saved by choosing energy-efficient data centers, and the CO_2 emissions can be reduced up to 60 times by placing VMs on data centers provided with green energy sources.

Keywords

Application component placement, cloud broker, cloud computing, cloud federation, green networking, meta-scheduling, tabu search, virtual machine placement.

4.1 Introduction

In today's ever changing communications environment, cloud computing facilitates the rapid deployment of new applications. The idea is to have a very flexible system that provides the required quality of service (QoS) to the application, and, therefore, a good quality of experience (QoE) to the user, while minimizing the resources involved in the deployment such as cost, energy consumption, and carbon footprint impact. In order to reduce cost and energy consumption, data center physical servers are assigned a number of Virtual Machines (VMs) on which to execute the applications.

The nature of communication applications is increasingly dynamic. In fact, applications are created, change their size, and are removed sometimes in very short lapses of time. In other words, they come, go and change on the fly. In such a context, for the effective management of a cloud computing network, it is important that the assignment of VMs to the coming applications presents the same type of dynamism, so that cloud computing resources are utilized in the most efficient manner.

The increasing number of cloud data centers allow service providers to rely on applications deployed across multiple data centers from one ore more cloud providers. For instance, a service provider can install VMs in an Amazon's data center in the US, some VMs in a European data center of the same infrastructure provider, and more VMs in a RackSpace data center in Asia. Counting on multiple data centers and providers improves the reliability of the system in case of failures, reduces communication delay by getting applications and data close to users, and reduces cost by taking advantage of daily variations in VM prices.

Another advantage of choosing among multiple data centers is to place VMs in data centers with high energy efficiency and green energy sources given the rising concerns about global warming. Studies show that data centers have a great impact on the world energy consumption, reaching up to 2% in 2007 and increasing at a 12% rate. Figure 4.1 shows that the power consumption of telecommunication networks and data centers together would rank as the 5^{th} country in the world [1]. However, not all the data centers have the same impact on the environment. The Power Usage Effectiveness (PUE) is the ratio of total power consumed by a data center divided by the power effectively consumed by the IT equipment. There exist data centers with a PUE of 2 (100% overhead): for each watt of IT equipment another watt is used for cooling and power distribution. On the other hand,

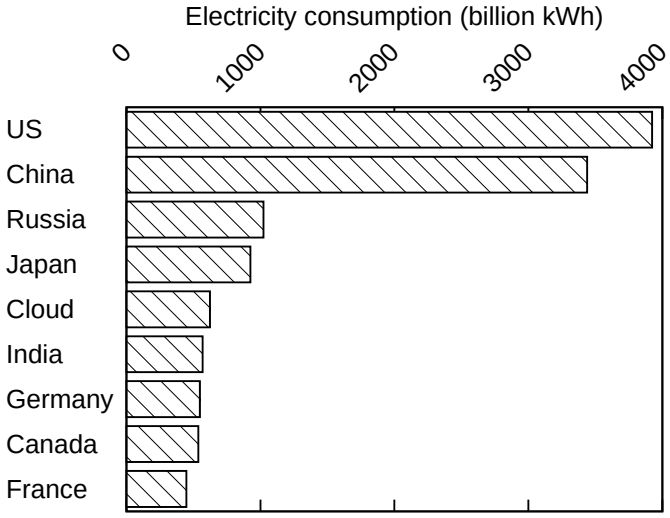

Figure 4.1 Energy consumption of countries and the cloud. Adapted from Cook [1].

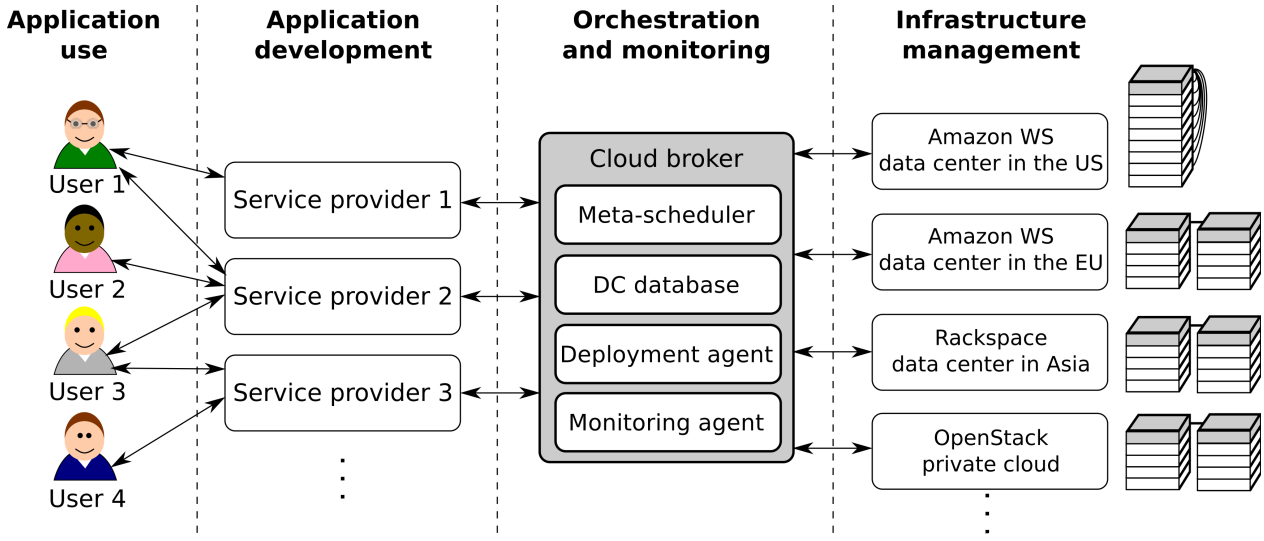

Figure 4.2 Actors and components interacting in the life-cycle of cloud applications.

there are data centers designed to get cold air from the environment, remove dust, and humidify it to drastically reduce the cooling equipment consumption. These data centers can achieve a PUE of 1.08, which represents an overhead of only 8% [25]. Besides the energy efficiency, the type of electricity generation process has a huge impact on CO_2 emissions. For instance, hydroelectric energy introduces 10 grams of CO_2 per kWh, and coal produces 960 g/KWh [11]. Studies show that 2% of the global CO_2 emissions are due to the Information and Communications Technology (ICT) industry, equivalent to the emissions of the aviation industry [82]. Choosing data centers with green energy sources can greatly reduce these emissions.

Using multiple cloud providers raises interoperability issues, and using a cloud broker is a way to solve them. Even if cloud computing services offer to dynamically add and remove VMs, different providers use different Application Programming Interfaces (APIs) to orchestrate the application life-cycle. Thus, a common platform to manage applications on multiple providers is strongly needed to simplify deployment and monitoring tasks. That platform is often called a cloud broker. Commercial cases include RightScale, Enstratius, Gravitant, and Scalr.

Cloud brokers provide software architects freedom to choose the desired data centers to place VMs, but they lack an optimization engine, or *meta-scheduler*, that helps make the right decisions. This paper proposes a dynamic optimization problem and an efficient algorithm that can be used as a meta-scheduler to place VMs in the right cloud data centers at the right time regarding costs, QoS, energy consumption, and CO_2 emissions. Figure 4.2 shows the whole picture about how actors interact in this context. Service providers develop applications that need to be deployed on the cloud. The cloud broker offers a platform to define each application architecture and the type of VMs needed. The cloud broker has a database of cloud providers with data center locations, QoS, and power consumption. The application configuration, database, and architect priorities are fed to the meta-scheduler that provides a solution with a data center for each VM. The user checks the solution costs, QoS, energy consumption, and CO_2 emissions calculated by the meta-scheduler, and if the solution is approved, the cloud broker starts the VMs in the chosen data centers. The cloud broker then keeps monitoring availability and QoS of the system.

The core of the optimization engine proposed in this article is implemented as a tabu search algorithm that selects a data center for each VM yielding large improvements in QoS and scaling to networks with thousands of nodes and VMs. In fact, the optimization algorithm has a tight execution time constraint (< 1 s) because it would be embedded to a cloud broker interactive system. The software architect gets an optimal solution, modifies priorities, and executes the optimization again to get a new solution in real time. In this

paper, the algorithm is embedded into a simulation procedure to allow the analysis of cloud networks and applications with respect to traffic, cost, and energy consumption.

The rest of the article is structured as follows. Section 4.2 presents the work that is the closest to ours. Section 4.3 describes the problem and a mixed integer programming model. The solution approach is clearly described in Section 4.4. The topology of networks and applications as well as all the parameters to evaluate our approach are presented in Section 4.5. The impact of the meta-scheduler on QoS, power consumption, CO_2 emissions, and cost is analyzed in Section 4.6. Finally, conclusions are presented in Section 4.7.

4.2 Related work

In the last years, cloud computing has attracted the attention of academia and industry to provide computing in an on-demand and pay as you use basis as an utility [8]. This model is so successful that the number of servers in the world is expected to grow by a factor of 10 between 2013 and 2020, and the information managed by enterprise data centers will grow by a factor of 14 [10]. The visions that the Internet pioneers had in 1969 are indeed happening [6]. The ability to request resources on-demand gave rise to proposals for combining clouds in a coordinated way. Buyya et al. [8] proposed cloud exchanges and markets where resources can be offered by infrastructure providers and requested by other service or infrastructure providers. Based on standards, dynamic algorithms handle transactions, define prices, and give consumers the required QoS. Ardagna et al. [83] proposed MODA-Clouds, a model driven approach for the design and execution of applications on multiple clouds. That approach is from the point of view of application developers whose high level requirements are translated into a programming code schema. Also, a decision support system is used to choose cloud providers taking into account availability, risks, and costs. The OPTIMIS approach proposed in Ferrer et al. [84] is a dynamic provisioning system to handle the whole service life-cycle from development, deployment, and operation in multi cloud architectures. Rochwerger et al. [85] defined the Reservoir model, an architecture for open federated cloud computing. That paper defines protocols and standards allowing cloud providers and consumers to exchange resources and deploy applications in a dynamic way. OpenNebula [86] is a system to manage a local cloud coordinated with remote clouds, such as Amazon Web Services and ElasticHosts. Commercial companies—e.g., RightScale, Enstratius, Gravitant, and Scalr—also started to offer cloud brokerage services as web platforms to deploy and monitor virtual machines across multiple cloud providers.

In the context of multi cloud architectures, meta-scheduling is the process of requesting resources from independently managed data centers. Leal et al. [17] proposed methods to

schedule tasks for grid computing, the scientific predecessor of cloud computing that is focused on High Performance Computing (HPC). The objective of these methods was to reduce the execution time of the HPC tasks. Two connected grids with 2013 processing elements were simulated using workloads from the Large Hadron Collider experiment. Chaisiri et al. [16] proposed stochastic integer programming models for placing VMs across multiple cloud providers. The objective was to take advantage of the price difference between reserved plans and on-demand plans. Reserved plans are cheaper but require to be acquired in advance, and the price of on-demand plan varies. The model handles uncertainty on future demands and prices. Tordsson et al. [19] proposed an integer programming model to place VMs across multiple providers and maximize the QoS. Each VM type in each cloud has a capacity, e.g., in requests per second. The model chooses a data center and a type of each VM in order to maximize the capacity subject to system and budget constraints. The meta-scheduling algorithm was tested with an HPC application across three cloud data centers and showed that using multiple cloud data centers improves performance and cost.

The approach presented in this paper includes a meta-scheduling algorithm to assign one data center to each VM. Then, different VM placement algorithms can be executed within each data center to assign a particular server to each VM. The following papers addressed the scheduling problem within the data center. Speitkamp and Bichler [52] proposed Mixed Integer Linear Programming (MILP) models to minimize the number of servers used by VMs. A branch and bound algorithm and heuristics were used to solve that problem. Srikantaiah et al. [3] analyzed trade-offs between energy consumption and performance and found optimal levels of resource utilization that minimize the average energy per request. Beloglazov et al. [53] proposed online heuristics to minimize server energy consumption considering the possibility to migrate VMs between servers. Addis et al. [57] proposed a decentralized heuristic to dynamically change server voltage and frequency and place VMs of multi-tier applications. The objective was to minimize response time, migration costs, and to maximize the infrastructure provider revenue.

Some authors also considered information sent between VMs to assign VMs to servers. For that purpose, Guo et al. [21] proposed the concept of Virtual Data Centers (VDCs). Each VDC requires a set of Virtual Machines (VMs) and there is a matrix that defines the amount of traffic between each pair of VMs. A similar approach was developed by Meng et al. [20]. The VMs that exchange most traffic are grouped into clusters, and nearby servers in the network are grouped into clusters as well. Then, each VM cluster is matched with a server cluster to improve the inter-VM communication.

With respect to our previous work, in Larumbe and Sansò [71] we proposed a network planning problem to define both the locations of data centers and software components along

with the information routing. The objective was to minimize the average delay of the information carried through the network. The problem was modeled as a MILP model and solved with AMPL-CPLEX. Then, in Larumbe and Sansò [31] we created a new model that considers energy and pollution issues. In that case, a multi-objective function was used to make trade-offs between different aspects of the problem: delay, cost, energy consumption, and CO_2 emissions. In Larumbe and Sansò [23], an efficient tabu search heuristic was proposed to solve the cloud network planning problem for large cases.

Differently from these references, this paper presents the following original contributions:

1. The proposal of a new online problem for VM placement across multiple cloud data centers considering cost, communication delay, power consumption, and CO_2 emissions.
2. Variations of each application workload over time are used to dynamically change the number of VMs, reducing costs, power consumption, and CO_2 emissions.
3. Special consideration of the communication delay between VMs and users, as well as between VMs themselves.
4. An extremely efficient heuristic that finds near optimal solutions in real time.

4.3 Problem description

We first present the network and application topology and the meta-scheduler definitions. Then, all the parameters referred to communication traffic, VMs, servers and costs are detailed. We continue with the system variables, the objective function, and finally the constraints.

4.3.1 Network topology, application, and meta-scheduler

A cloud computing network is represented as a graph $G^N(V^N, A^N)$ composed of access nodes $\mathcal{A}$, backbone routers $\mathcal{B}$, data centers $\mathcal{D}$, and the links that connect these nodes. Each access node aggregates a set of users that can be an organization, an Internet Service Provider (ISP), an Internet Exchange Point (IXP), or a city. The data centers, from one or multiple providers, host servers with the ability to execute VMs. Each backbone router forwards traffic received from its incident links. If the complete network topology is not known, A^N contains direct virtual links with the only requirement of measured end-to-end delay. For instance, the delay between two data centers of different providers can be measured through ping ICMP packets even if the whole Internet path is not known. The model is kept general enough for the case that the whole backbone network is known.

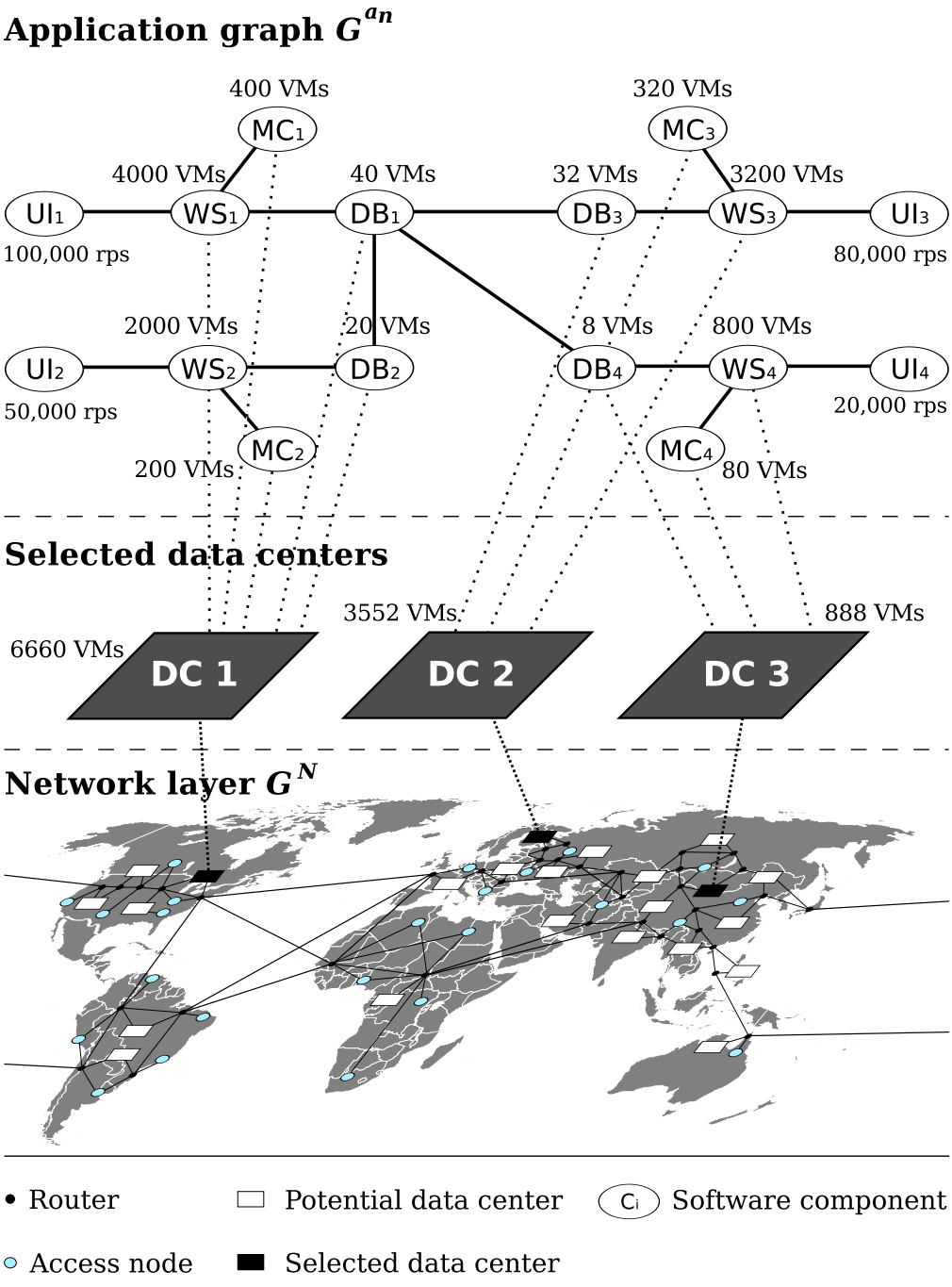

Figure 4.3 Application and network mapping.

A module called *meta-scheduler* receives requests to add and remove applications. Let a_n represent a particular application and $G^{a_n}(V^{a_n}, A^{a_n})$ a graph that represents that application architecture. Each node in V^{a_n} is a software component, and each arc in A^{a_n} from software component i to software component j represents the fact that i sends information to j. Each software component has a specific purpose in the application and can be executed in one or multiple VMs or in the user computers and mobile devices. The number of VMs required to run each software component at each period of the day is an input of the meta-scheduler based on workload forecasts given by the software architects that use the cloud broker [41, 63].

Figure 4.3 shows an example of a multi-tier application graph inspired by the most popular social network, Facebook [50]. In this case, the application is composed of four types of software components: User Interface (UI), Web Service (WS), Memcache (MC), and Database (DB). The UI handles the interaction with users and it is executed in each web browser as a Javascript/HTML5 application. To make the representation of the application graph scalable, there is one UI for each access node, e.g., for each city. That UI represents the aggregation of all the HTTP requests received from that access node. The WS handles each HTTP request from the UI, queries required information, processes it, and gives an HTTP response. The WS is executed in multiple VMs to achieve high throughput. In this example, each VM can handle 25 requests per second (rps) with the required QoS. For instance, UI_1 sends 100,000 rps, thus 4,000 VMs are required by WS_1 to process them. The WS gets information from the DB, which stores it on hard disks. The WS sends that information to a MC that stores it in memory with much lower latency than the DB. In this example, for each 100 VMs of WS, there are 10 MC VMs, and 1 DB VM.

$G_t^A(V_t^A, A_t^A)$ is the graph that contains all the applications that have been already deployed and are being executed at time t. The nodes in V_t^A are all the software components and the arcs in A_t^A represent communication demands between software components. If the system receives a request to add a new application a_n at time $t+1$ then:

$$G_{t+1}^A = (V_t^A \cup V^{a_n}, A_t^A \cup A^{a_n})$$

If a request to remove an application a_n is received at time $t+1$ then:

$$G_{t+1}^A = (V_t^A - V^{a_n}, A_t^A - A^{a_n})$$

The set P^{a_n} is a parameter that denotes possible or mandatory assignments of software components to network nodes for application a_n. This is used to restrict which nodes—data centers or access nodes—can host each software component. For instance, the UI software components of Figure 4.3 must be placed in their respective access nodes. Software compo-

nents WS_1, MC_1, and DB_1 could be placed in anyone of the data centers. The set P^{a_n} can also be used to restrict a subset of the data centers for a software component. For instance, WS_4, MC_4, and DB_4 could be constrained to be in data centers located in Asia because of privacy legislation.

The result of the optimization problem solved by the meta-sheduler is a mapping M_{th}: $V_t^A \rightarrow V^N$ that specifies which node—data center or access node—is assigned to each software component in the h'th period of the day for the applications available in the system at time t. That reflects the online and dynamic aspects of the model. It is online because at each time t a new application can be added or removed. It is dynamic because the size of the application workload changes over the day and there is a potentially different mapping for each period of the day h. Then the meta-scheduler has to define the mapping M_{th} for each period keeping the mapping of previously deployed applications.

To sum up, the parameters regarding the network topology, applications, and meta-scheduler are:

$G^N(V^N, A^N)$
: Network topology with data centers, access nodes, and routers.

$G^{a_n}(V^{a_n}, A^{a_n})$
: Graph of software components and communication demands of application a_n.

$G_t^A(V_t^A, A_t^A)$
: Graph of software components and communication demands of all the applications executing at time t.

P^{a_n} Possible assignments of software components to nodes.

$\mathcal{H}$ Set of indexes for the periods in the day.

When a new application must be deployed at time t, an optimization problem must be solved to decide the location of each software component in each period of the day. The result is:

M_{th} Mapping of software components to nodes given as a result of the meta-scheduler optimization solution for applications in G_t^A and period $h \in \mathcal{H}$.

Now that the network topology, each application architecture, and the basic dynamics of the meta-scheduler are defined, we proceed to define the remaining parameters. These constants, system variables, objective function, and constraints define the optimization problem that is solved to add a new application at time t knowing the current state of the network.

4.3.2 Communication traffic

The software components communicate between them by sending and receiving packets of information through the network. Each packet that goes from a node i to a node j follows a path in the network composed of routers and links. Each step in the path adds processing, queuing, transmission, and propagation delay. The following are the parameters regarding the communication traffic:

b_{dh} Expected throughput of demand $d \in A_t^A$ in period $h \in \mathcal{H}$ (in bps).

τ_{eh} Delay of link $e \in A^N$ in period $h \in \mathcal{H}$ (in seconds per packet).

$\overline{\tau_d}$ Maximum allowed delay for demand $d \in A^A$ (in seconds per packet).

4.3.3 VMs and dynamic scaling

There are different types of VMs depending on the application requirements. For instance, Amazon EC2 provides small VM instances with 1 EC2 Compute Unit and 1.7 GB memory, medium VM instances with 2 EC2 Compute Units and 3.75 GB memory, and large VM instances with 4 EC2 Compute Units and 7.5 GB memory [39]. An EC2 Compute Unit is the capacity of a 1 GHz 2007 Xeon processor. Each VM uses a fraction of the server resources, so each server can host a maximum number of VMs.

The number of VMs required by each software component varies in each period of the day, a feature called dynamic scaling or auto scaling. At peak time, a software component will require a large number of VMs to handle a high workload. At other periods of the day, the number of VMs can be reduced. The required VMs for each period are defined with forecast models based on the application workload history [41, 63]. That elasticity allows a good use of data center resources and power consumption. The meta-scheduler solution defines in which data centers these VMs are placed in each period.

Depending on the type of application, VMs could be started in one data center in a period of the day, and be later removed from that data center and started again in another data center. The meta-scheduler can produce that kind of dynamic plan to take advantage of reduced overnight prices for spot instances. Then a deployment module starts and removes VMs in the specified data centers in each period of the day.

The VMs and servers are defined by the following parameters:

$\mathcal{T}$ Set of VM types, e.g., 1 = small instance, 2 = medium instance, 3 = large instance, 4 = high RAM instance, 5 = high CPU instance, 6 = small spot instance, 7 = storage volume.

r^1_{imh} Number of VMs of type $m \in \mathcal{T}$ required by software component i in period $h \in \mathcal{H}$.

r^2_m Number of VMs of type $m \in \mathcal{T}$ that can be hosted in a server.

r^3_k Capacity of data center $k \in \mathcal{D}$ in number of servers.

m_i 1 if software component $i \in V^{a_n}$ can be migrated during the day; 0 otherwise.

4.3.4 Cost

A cloud broker allows service providers to optimize VM and communication costs paid to infrastructure providers. Some VM types could keep a constant price during the day, and others could be dynamically changed by infrastructure providers. That is the case of the so-called spot instances whose price varies depending on the residual data center capacity. A data center that has spare capacity overnight can reduce the spot instance prices to attract more customers. This model takes advantage of dynamic prices to optimize service provider costs and keep the infrastructure highly utilized. These costs are defined as:

c^1_{kmh} Price of a VM of type $m \in \mathcal{T}$ in data center $k \in \mathcal{D}$ in period $h \in \mathcal{H}$ (in \$ / VM / hour).

c^2_{eh} Cost for sending traffic through link $e \in A^N$ in period $h \in \mathcal{H}$ (in \$ / GB).

4.3.5 Energy and environment

Each data center has a particular design and location that defines its Power Usage Effectiveness (PUE), which is the ratio of total power—of IT equipment, cooling, and power distribution—divided by the IT equipment power. The software architect that needs to deploy a new application can restrict the amount of power consumption in order to optimize it. In that case, the meta-scheduler will choose data centers with appropriate power efficiency. In the same way, depending on the type of energy sources that provides a data center, the amount of greenhouse gas emissions can be calculated [1]. For instance, hydroelectric power introduces 10 g CO_2 / kWh, and coal produces 960 g CO_2 / kWh [11]. The planners can also limit that factor for each new application. The following are the parameters of power consumption and CO_2 emissions:

w^1_k Average power consumption of a server in data center $k \in \mathcal{D}$ (in W).

w_k^2 PUE of data center k, i.e., ratio of total power over IT equipment power.

$w_{a_n}^3$ Power limit for application a_n (in W).

$w_{a_n}^4$ Limit to the PUE of application a_n.

w_k^5 Emissions of CO_2 for the energy used by data center k (in g/Wh).

$w_{a_n}^6$ Limit to the CO_2 emissions of application a_n (in g/h).

$w_{a_n}^7$ Limit to the CO_2 emissions per energy consumed by application a_n (in g/kWh).

4.3.6 System variables

The model variables keep track of the mapping between software components and nodes, and the traffic in each link. When a new application is requested, the meta-scheduler optimizes this model to define the mapping of the new application, that is in which data center each software component will be placed in each period of the day. The routing is simultaneously solved to minimize traffic cost and constraint communication delay. The following are the system variables:

x_{ikh} 1 if software component $i \in V_t^A$ is placed in node $k \in V^N$ in period $h \in \mathcal{H}$; 0 otherwise.

f_{deh} Amount of communication demand $d \in A_t^A$ transmitted by link $e \in A^N$ (in bps) in period $h \in \mathcal{H}$.

4.3.7 Objective function

The objective function to minimize is the daily cost of VMs and traffic for the new application a_n. The expected workload, number of VMs, and communication traffic will change in each period of the day, thus the formula adds the cost paid in each period to calculate the whole daily cost:

$$z = \sum_{h\in\mathcal{H}} \sum_{k\in\mathcal{D}} \sum_{m\in\mathcal{T}} \sum_{i\in V^{a_n}} \frac{24}{|H|} c_{kmh}^1 r_{imh}^1 x_{ikh} + \sum_{h\in\mathcal{H}} \sum_{e\in A^N} \sum_{d\in A^{a_n}} \frac{24}{|H|} c_{eh}^2 \frac{3600}{8\cdot 10^9} f_{deh} \tag{4.1}$$

4.3.8 Constraints

• *Software component placement*

$$\sum_{(i,k)\in P^{a_n}} x_{ikh} = 1 \qquad \forall \quad i \in V^{a_n}, h \in \mathcal{H} \tag{4.2}$$

$$\sum_{\substack{k\in V^N/\\ (i,k)\notin P^{a_n}}} x_{ikh} = 0 \qquad \forall \quad i \in V^{a_n}, h \in \mathcal{H} \tag{4.3}$$

$$x_{ikh} = 1 \qquad \forall \quad h \in \mathcal{H}, (i,k) \in M_{t-1,h} \tag{4.4}$$

$$x_{ikh} = 0 \qquad \forall \quad i \in V^A_{t-1}, k \in V^N, h \in \mathcal{H}, \; (i,k) \notin M_{t-1,h} \tag{4.5}$$

Equations (4.2) and (4.3) state that each software component of the new application must be placed in exactly one node of the network—data center or access node—, and that assignment of software component to node must be in the set P^{a_n} to be feasible. Equations (4.4) and (4.5) keep the previous applications fixed in their already allocated nodes.

• *Migration of software components*

$$x_{ikh} - x_{ik,h+1} \leq 0 \qquad \forall \quad i \in V^{a_n}/m_i = 0 \quad \forall \quad k \in \mathcal{D}, h \in \mathcal{H} \tag{4.6}$$

Constraint (4.6) states that if software component i was hosted in data center k in period h, then it must remain in the same data center in the next period, for all components that cannot be migrated.

• *Flow conservation*

$$\sum_{e\in\Gamma^-_k} f_{deh} + x_{ikh} b_{dh} = \sum_{e\in\Gamma^+_k} f_{deh} + x_{jkh} b_{dh} \qquad \forall d = (i,j) \in A^A_t, k \in V^N, h \in \mathcal{H} \tag{4.7}$$

This family of constraints relates the location of software components and the routing of traffic demands. It associates the application and network layers. For a demand from a software component $i \in V^A_t$ located in node $k_1 \in V^N$ to a software component $j \in V^A_t$ located in node $k_2 \in V^N$, the flow of traffic b_d starts in k_1 and ends in k_2. In each intermediate node k between k_1 and k_2, the flow is conserved.

- *Data center capacity*

$$\sum_{i \in V_t^A} \sum_{m \in \mathcal{T}} \frac{r^1_{imh}}{r^2_m} x_{ikh} \leq r^3_k \qquad \forall \quad k \in \mathcal{D}, h \in \mathcal{H} \tag{4.8}$$

The left part of Constraint (4.8) approximates the number of servers used by VMs of software components placed in data center k and period h. That number is bounded by the physical capacity of the data center.

- *Delay*

$$\sum_{e \in A^N} \tau_{eh} \frac{f_{deh}}{b_{dh}} \leq \overline{\tau_d} \qquad \forall \quad h \in \mathcal{H}, d \in \mathcal{A}^A_t \tag{4.9}$$

Constraint (4.9) limits the delay of each communication demand. That can be used to achieve a desired QoS by placing VMs in data centers that are close to access nodes. It can also be used to specify that two software components must be placed in the same data center.

- *Energy and environment*

$$\sum_{\substack{k \in V^N \\ i \in V^{a_n} \\ m \in \mathcal{T}}} \frac{r^1_{imh}}{r^2_m} w^1_k x_{ikh} w^2_k \leq w^3_{a_n} \qquad \forall \quad h \in \mathcal{H} \tag{4.10}$$

$$\sum_{\substack{k \in V^N \\ i \in V^{a_n} \\ m \in \mathcal{T}}} \frac{r^1_{imh}}{r^2_m} w^1_k x_{ikh} w^2_k w^5_k \leq w^6_{a_n} \qquad \forall \quad h \in \mathcal{H} \tag{4.11}$$

Constraint (4.10) limits the power consumption of the new application by considering the PUE of each data center in each period of the day. In a similar way, Constraint (4.11) puts a limit to the CO_2 emissions.

Because applications have different sizes, setting a limit to the power consumption in absolute values as in Constraints (4.10) is in practice difficult. An alternative way is to limit the overhead introduced by data center equipment in relative terms. To do that, we define the application PUE as the sum of total power consumed by the application including data center overhead, divided by the power consumed by the servers that host the application's VMs. Then, that value can be bounded by an expected PUE. A similar approach can be considered for the CO_2 emission constraint. Constraints (4.12) and (4.13) limit the power consumption and CO_2 emissions in relative terms, respectively. Note that the denominators can be passed to the right terms to keep the model linear.

$$\frac{\sum_{\substack{k\in V^N \\ i\in V^{a_n} \\ m\in\mathcal{T}}} \frac{r^1_{imh}}{r^2_m} w^1_k w^2_k x_{ikh}}{\sum_{\substack{k\in V^N \\ i\in V^{a_n} \\ m\in\mathcal{T}}} \frac{r^1_{imh}}{r^2_m} w^1_k x_{ikh}} \leq w^4_{a_n} \qquad \forall \quad h \in \mathcal{H} \tag{4.12}$$

$$\frac{\sum_{\substack{k\in V^N \\ i\in V^{a_n} \\ m\in\mathcal{T}}} \frac{r^1_{imh}}{r^2_m} w^1_k w^2_k w^5_k x_{ikh}}{\sum_{\substack{k\in V^N \\ i\in V^{a_n} \\ m\in\mathcal{T}}} \frac{r^1_{imh}}{r^2_m} w^1_k w^2_k x_{ikh}} \leq w^7_{a_n} \qquad \forall \quad h \in \mathcal{H} \tag{4.13}$$

- *Domain of variables*

$$x_{ikh} \in \{0,1\} \qquad \forall \quad i \in V^A_t, k \in V^N, h \in \mathcal{H} \tag{4.14}$$

$$f_{deh} \in \mathbb{R}_{\geq 0} \qquad \forall \quad d \in A^A_t, e \in A^N, h \in \mathcal{H} \tag{4.15}$$

Finally, we define the domain of each variable. Assignment variables are binary, and flow variables are non-negative real numbers.

4.4 Solution approach

The meta-scheduler must be integrated in an interactive system that allows service providers to get solutions in real time. A planner that needs to deploy a new application specifies application requirements and constraints on QoS, power, and CO_2 emissions and get the placement of software components, and consequently VMs, with optimal cost. To make such interactive system the meta-scheduler should give the solution in a short execution time, e.g., in less than one second.

In this context, we propose a very efficient tabu search heuristic [13, 73] that solves the MILP model. It is inspired in the algorithm that we proposed for the location of data centers and software components that achieved excellent optimality gaps and execution times [23]. The new algorithm must provide two fundamental features to the meta-scheduler: receive consecutive applications online and handle workloads that change over time.

The online aspect is solved by keeping the graph of applications that were already deployed (G^A_t) in memory, calculate data center residual capacities, and place VMs of the new application. The multi-period aspect is solved by getting an independent mapping (M_{th}) for each period of the day based on the workload of that period.

As can be seen in Algorithm 2, the general structure of the meta-scheduler is carried by the following functions:

Algorithm 2 Add or remove an application to a solution.

function SCHEDULE(*Request*, M)
  for each period h in H **do**
    SCHEDULEINPERIOD(*Request*, M, h)
  end for
end function

function SCHEDULEINPERIOD(*Request*, M, h)
  $a_n \leftarrow$ *Request* application
  if *Request* type is add **then**
    if $h > 0$ **then**
      Add to P^{a_n} components already placed in M_0 and that cannot be migrated.
    end if
    $M_h \leftarrow$ INITIALGREEDYSOLUTION(a_n, M_h)
    $M_h \leftarrow$ TABUSEARCH(a_n, M_h)
  end if
  if *Request* type is remove **then**
    Remove a_n's software components from M_h
  end if
end function

function INITIALGREEDYSOLUTION(a_n, S)
  for each component c in a_n **do**
    Move c to the node that least increases the objective function in S
  end for
  return S
end function

function TABUSEARCH(a_n, *Current*)
  $S \leftarrow$ *Current*
  Best $\leftarrow$ *Current*
  $L \leftarrow \{\}$
  $i \leftarrow 0$
  repeat
    Choose the pair $(c, d) \in P^{a_n} - L$ that minimizes z when moving software component c to d in solution S.
    Move c to d in S
    Add (c, d) to tabu list L for q iterations.
    $i \leftarrow i + 1$
    if $z(S) < z($*Best*$)$ **then**
      Best $\leftarrow S$
      $i \leftarrow 0$
    end if
  until $i =$ *MAX_ITERATIONS*
  return *Best*
end function

Schedule

This is the main function that schedules a new request to add or remove an application to the system. The parameters are the new request and the current mappings of applications in the network. There is one mapping for each period of the day ($M_0, \ldots, M_{|H|-1}$)), and the request is solved for each one of them.

ScheduleInPeriod

The VMs are placed based on the workload of a specific period of the day. VMs that cannot be migrated are kept in the data centers chosen in solution M_0. The first step to add an application is to place software components using a greedy algorithm.

InitialGreedySolution

When a software component is added to a node, the objective function is increased because of VM and traffic costs. Each component is placed in the node that least increases the objective function.

TabuSearch

The initial solution might be sub optimal. Then a tabu search heuristic moves the new software components between data centers to reduce the objective function. If the objective function does not decrease for a number of iterations, the algorithm stops.

More details on the fine tuning of tabu search heuristic for the placement of software components, practical implementation issues, and optimality gaps can be consulted in Larumbe and Sansò [23].

4.5 Case study

In this section, we study the characteristics of the cloud computing networks and applications and we define all the parameters of the model. This network will be used to simulate the placement of applications with the proposed meta-scheduling algorithm.

4.5.1 Network topology

The network G^N includes a set of access nodes $\mathcal{A}$, backbone routers $\mathcal{B}$, and data centers $\mathcal{D}$. The nodes are distributed across multiple locations around the globe. We considered the 500 largest cities in the world [75], the number of Internet users in each of them [76], and their geographic location [77]. Figure 4.4 shows a network with the largest 100 cities in the world

Figure 4.4 Network topology with the largest 100 cities in terms of Internet users. *Image generated with Google Maps API.*

in terms of Internet users produced by our generator. The bright circles are access nodes that represent cities, the black dots are backbone routers, and the white boxes are data centers.

4.5.2 Application topology

There is a set of applications that arrive consecutively. Each user component has a known location, one of the 500 cities. The meta-scheduler decides where to place the server components. The server components of an application are randomly connected to test that the meta-scheduler can handle any type of application. Each user component of the application is connected to one of the randomly chosen server components. These connections are communication demands that will be routed through the network when the components are placed. The size of the application graph depends on the amount of traffic handled by the application. The number of user components (u) is between 1 and 500, proportional to the total application traffic. The number of server components is randomly chosen between $\lceil u/2 \rceil$ and $\lfloor 1.5u \rfloor$. The number of VMs is also proportional to the application traffic. These VMs are uniformly distributed among the server components.

4.5.3 Communication traffic

The distribution of the application traffic was based on the traffic estimation of the Internet Autonomous Systems (ASs) [87] that follows a power law; that is, the largest ASs are responsible for most Internet traffic and many ASs produce little traffic. We used a Pareto distribution with parameters $\alpha = 1.1$ and $\beta = 8.5149$. For instance, 40,000 applications generated this way produce 4.3 Tbps of total traffic, 107 Mbps of average traffic per application, and the top 20% applications produce 88% of traffic.

90% of application traffic corresponds to demands between user and server components, and 10% are traffic demands between server components themselves. The total user component traffic is distributed among user components in such a way that the traffic of each user component is proportional to the number of Internet users in its corresponding city. The delay of each network link was set to 1 ms plus the propagation delay, which is calculated as the distance between the nodes divided by the speed of light in fiber (200,000 km/s).

4.5.4 Dynamic workload

The workload of each application changes over the day. There are periods of peak utilization and periods of low utilization. The workload peaks were established at 8 AM and 8 PM, taking into account the local time of the access node's city. Since the number of VMs is proportional to the application traffic in each period, then the number of VMs follows the same oscillating behavior. That was produced by using a sinusoid function with peaks that are 20% higher than the average workload.

4.5.5 Servers

In these cases, each server has 16 cores at a frequency of 2.4 Ghz and 80 GB of RAM. Each VM has allocated 800 Mhz and 1.6 GB RAM, allowing 48 VMs per server. Each data center has a predefined capacity, either 64,000 or 128,000 servers.

4.5.6 Power consumption and greenhouse gas emissions

The average power consumption of each server was set to 250 W. The PUE of the data centers is between 1.07 and 2, depending on the latitude of the data center (0° to 61° North and 0° to 61° South) to simulate data centers with different power efficiency. The amount of CO_2 introduced to the environment was randomly chosen between 10 (hydroelectric), 38 (geothermic), 66 (nuclear), 443 (natural gas), 778 (diesel), and 960 g / kWh (coal) [11].

4.5.7 Cost

VM prices are uniformly distributed between \$0.06 and \$0.18 / VM / hour, based on the fact that the price of a small Amazon EC2 instance was \$0.065 / VM / hour in 2013 [39]. The cost of sending information from a data center to the Internet is \$ 12 / GB, and from the Internet to a data center is free, based on that provider prices.

4.6 Results

4.6.1 Experience procedure

In order to study the behavior of the proposed meta-scheduler and the relation between QoS, power consumption, CO_2 emissions, and cost, we have implemented a simulator that produces consecutive requests. Each request adds a new application that must be placed by the meta-scheduler, or removes an existing one.

For each request, a random number was produced to decide if an application will be added or removed. The probability for adding was set to $\frac{2}{3}$ and for removing was set to $\frac{1}{3}$. If the request is to deploy a new application, then a new application is generated with the parameters of traffic and VMs seen in Section 4.5. The meta-scheduler is executed to decide the placement of the new application's VMs and the current mapping is updated with the new VMs in the data centers. If the new request is to remove an application, then a random application is chosen and its VMs are removed from the current mapping. The fact that the probability to deploy a new application is the double of the probability to remove an existing application makes an increasing data center utilization over time.

4.6.2 The set of experience

The network used for this set of experiences has 50 data centers, 500 access nodes, and 569 backbone routers; that is a total of 1119 nodes. 1,000 requests to add or remove an application were generated. The experience period represents a whole year of a cloud broker.

4.6.3 Setting

The tabu search was programmed in C++ and compiled in gcc 4.5.1. The simulations were performed on an Intel i7 with 4 cores at a frequency of 2.8 GHz.

4.6.4 Relation between delay, power, CO_2, and cost over time

The online aspect of the problem is analyzed in this section using applications that arrive consecutively during one year. The effectiveness of the method is proved by comparing an

unrestricted case where the cheapest data center is used for all the applications versus cases that constraint the QoS, power consumption, and/or CO_2 emissions.

Communication delay

The first results show the impact of communication delay constraint (4.9) when deploying applications. This aspect is fundamental to QoS because the communication delay is added to the VM processing time to make the total response time.

Figure 4.5 shows the average delay of the communication demands as applications are added to or removed from the system. One of the curves shows results without any delay constraint, and the other one, with a constraint to the minimal communication delay for each application. When no delay constraint is used, all the applications were placed in the less expensive data center giving an average delay of 93 ms. On the other hand, when delay is minimized, the meta-scheduler places each application in the data center that is closest to its users giving as a result an average delay of 16 ms, 6 times shorter than the first case.

Figure 4.6 shows the relation between delay and cost. Each chart point is the total cost of VMs and traffic of all the applications deployed at the end of the simulation using a specific delay bound. As the delay bound is shorter, the cost increases because feasible data centers are those that are closer to users, that might be more expensive. When the delay constraint is less tight, less expensive data centers can be chosen. For instance, restricting the communication delay to 30 ms (a round trip time of 60 ms) only costs 11% more than the unconstrained case.

Power consumption

Figure 4.7 shows power consumption over time when there is a power constraint, and when there is not. When a new application is deployed, power consumption increases because a larger number of servers are used. On the other hand, power consumption decreases when an application is removed because some servers are not used anymore. In this experience, the power consumption increases over time because the probability of adding an application is the double of removing one. When the power consumption constraint (4.12) is not used, all the applications were deployed on the cheapest data center that in this case has a PUE of 1.55. When Constraint (4.12) was used to choose the most energy-efficient data center, the applications were deployed on a data center with PUE 1.09. This way, 30% of power is saved thanks to choosing energy-efficient data centers.

Figure 4.8 shows the cost obtained with different power consumption bounds. As the power consumption constraint is relaxed, cost decreases because any data center can be

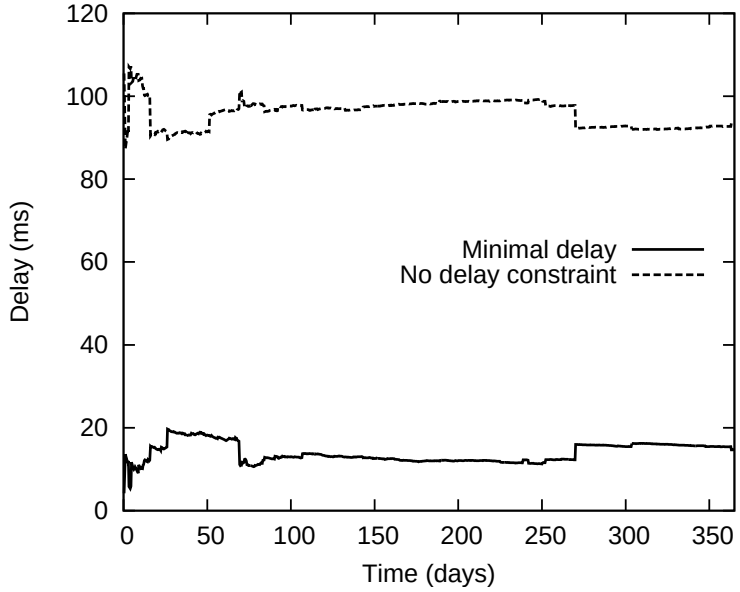

Figure 4.5 Delay over time.

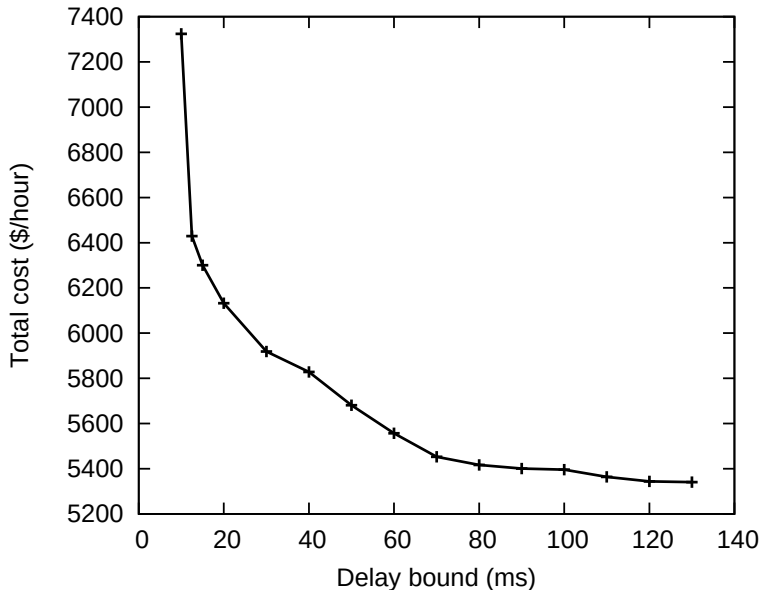

Figure 4.6 Trade-off between delay and cost.

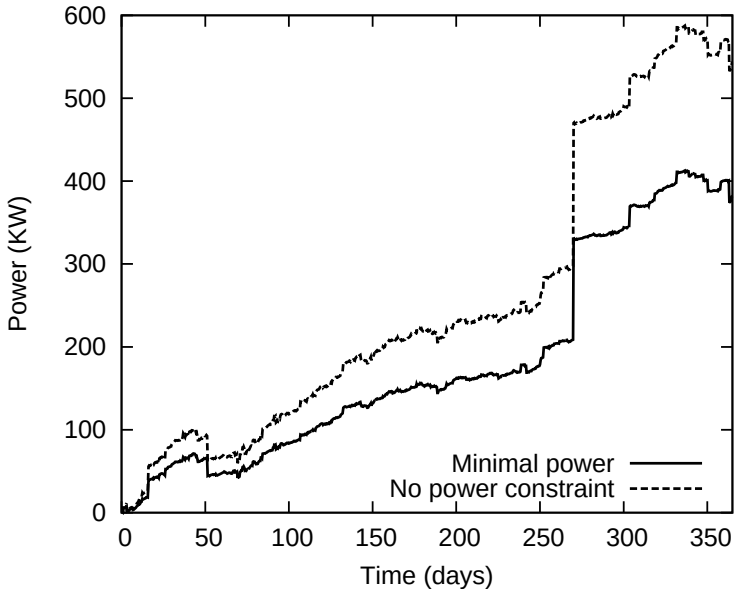

Figure 4.7 Power consumption over time.

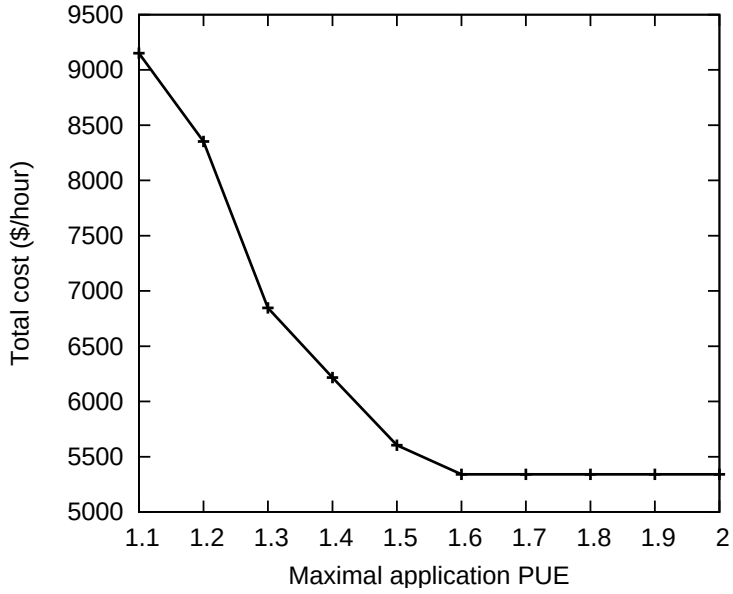

Figure 4.8 Trade-off between power consumption and cost.

choosen. When the limit to the application PUE is 1.6 or more, cost remains constant because the cheapest data center has PUE 1.55. When the restriction is set to use smallest PUEs, cost increases because some of them are more expensive. In this experience, the data center that most reduces power consumption costs almost the double of the least expensive. Nevertheless, real data centers could be both energy-efficient and provided with inexpensive electricity.

Greenhouse gas emissions

CO_2 emissions over time are presented in logarithmic scale in Figure 4.9. The same way as power consumption, CO_2 emissions increases over simulation time because more applications are added than removed. Constraining CO_2 emissions to the minimal value has an impact of several order of magnitude. That behavior can be explained by the fact that hydroelectric only produces 10 g CO_2 / kWh, and on the other hand, natural gas introduces 443 g CO_2 / kWh and coal, 960 g CO_2 / kWh. In fact, by the end of the simulation presented in Figure 4.9, 4 Kg of CO_2 were emitted in the last hour whereas in the non constrained case, 240 Kgs were emitted. This depends on the type of data center chosen to accommodate the application VMs.

Figure 4.10 shows each solution cost as a function of CO_2 emission constraint (4.13). When the limit is between 150 and 800 g CO_2 / kWh, the cost is kept around the optimal value because inexpensive data centers can achieve these values. When emissions are bound to 100 g/kWh, cost jumps 24% because data centers with more expensive VM prices are used. When Constraint (4.13) is set to produce the least possible amount of CO_2, the total cost is the double of the optimal value. This is because the price range assigned to VMs in this experience was between \$0.06 and \$0.18 / VM / hour.

4.6.5 Application workload during the day

The application workload varies during the day having periods of high throughput and periods of low throughput. Infrastructure providers allow service providers to create and destroy VMs on-demand to handle that workload in an efficient and economic way. The service providers have the advantage of paying for the VMs needed instead of paying for the peak capacity. Infrastructure providers can optimize power consumption by keeping on only the servers needed for the active VMs.

Figure 4.11 shows the power consumption consumed by the servers in one of the data centers chosen by the meta-scheduler during each period of the day. The lowest periods

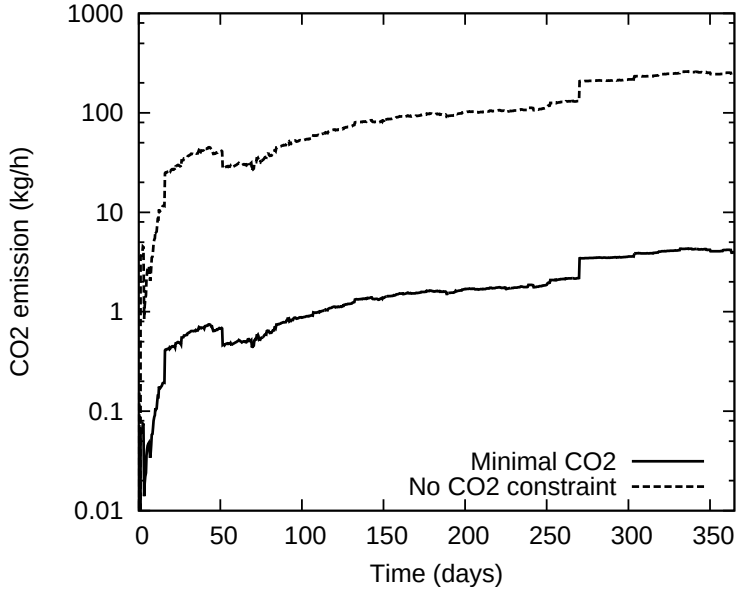

Figure 4.9 CO_2 emissions over time.

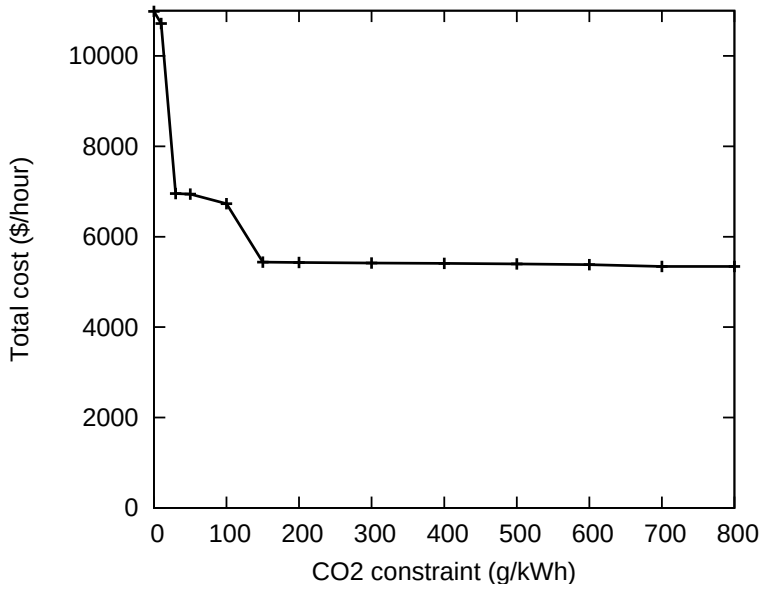

Figure 4.10 Trade-off between CO_2 emissions and cost.

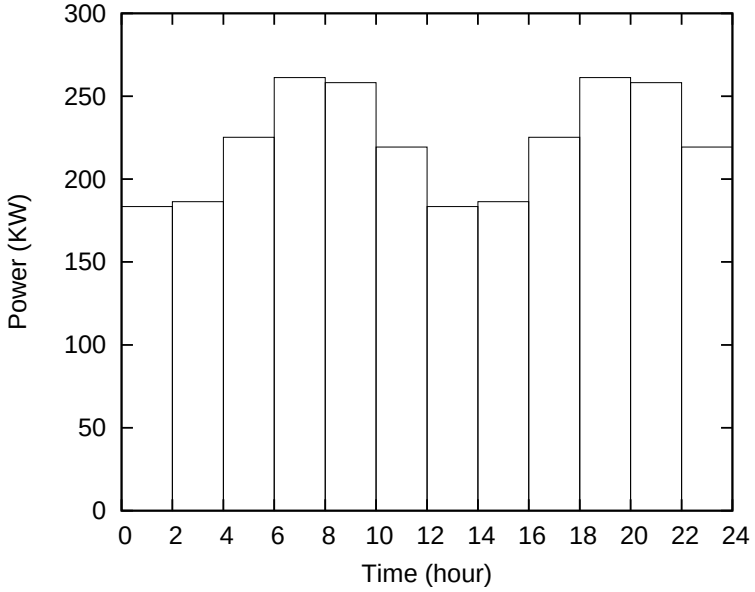

Figure 4.11 Data center power consumption.

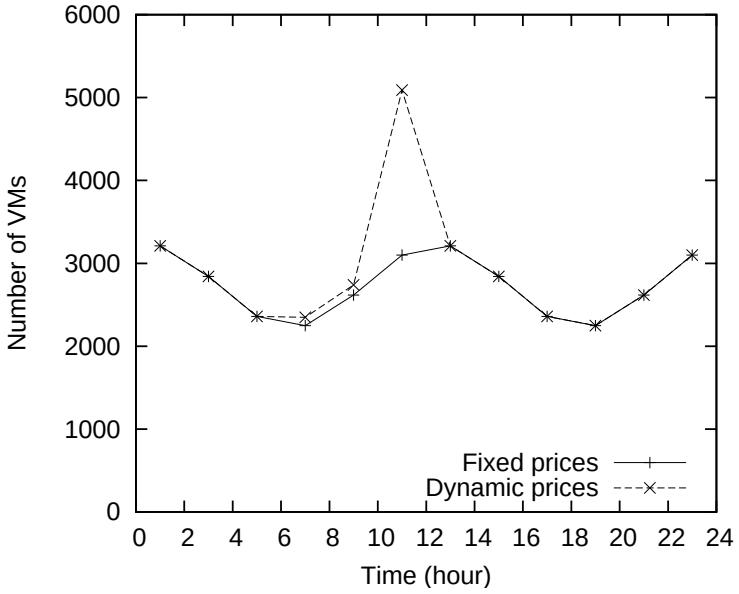

Figure 4.12 Spike in the number of VMs when prices are changed.

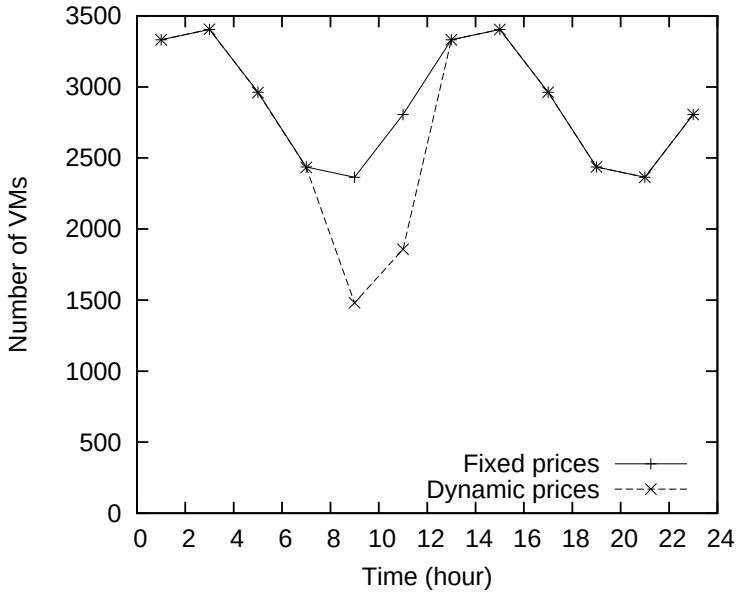

Figure 4.13 Low data center utilization when prices are changed.

consume 30% less power than the highest periods, highlighting the importance of considering dynamic workloads in the meta-scheduler.

The users and data centers considered in these experiences are geographically distributed around the globe. Each city has its time difference based on its geographic longitude. We explored the possibility of reducing 50% the VM prices in periods of low workload to attract users from other cities. The meta-scheduler effectively migrated software components to the data centers that reduced VM prices and reduced 5% the total cost paid by service providers. However, that produced very high spikes in the data center that reduced the prices (between 7:00 and 11:00 in Figure 4.12), and very low utilization in the data center that had the VMs in the first place (between 9:00 and 11:00 in Figure 4.13).

Thus, we conclude that considering dynamic workload is very important both for service provider costs and infrastructure provider power consumption. On the other hand, simply reducing the VM price to the half at night reported small benefits to service providers at expenses of very spiky data center workloads. Other pricing schemes should be studied to improve that behaviour.

4.6.6 Execution time

The efficiency of the meta-scheduler can also be seen in terms of the execution time. To study the scalability of the method, we evaluated the average execution time that the meta-scheduler takes to solve one request—adding or removing an application—using different

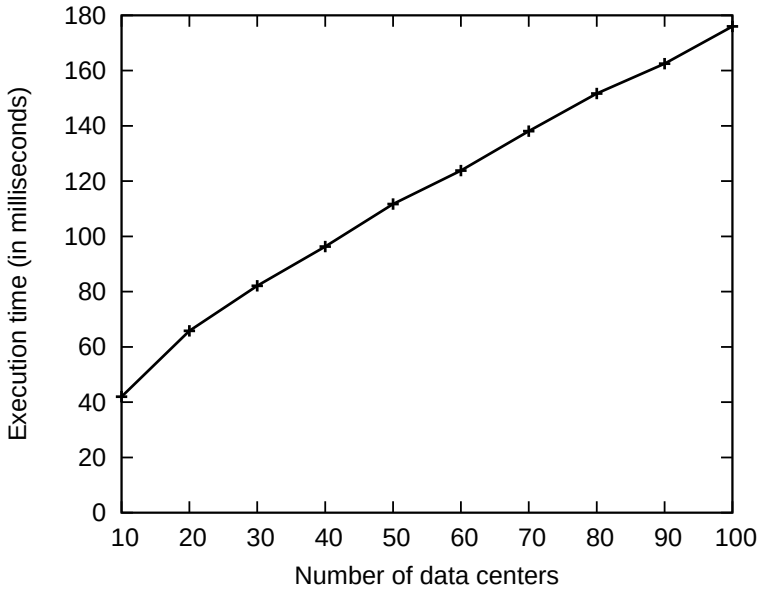

Figure 4.14 Average meta-scheduler execution time over the network size.

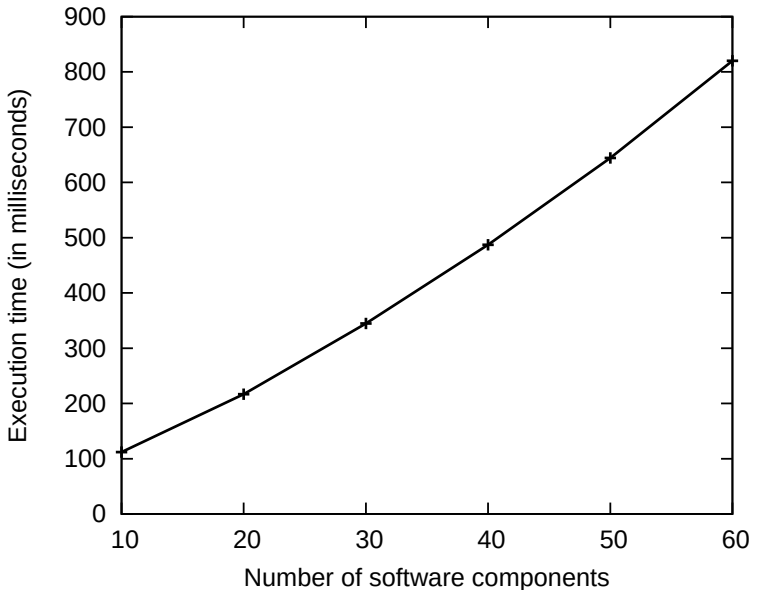

Figure 4.15 Average meta-scheduler execution time over the application size.

network and application sizes. Figure 4.14 shows the request solving time for different numbers of data centers, ranging from 10 to 100. Each point is the result of a simulation where 100 applications where added to or removed from the network. Each application had 10 software components with random traffic demands between them as described above. Figure 4.15 shows the variation of the request solving time over the number of software components in each application, ranging from 10 to 60 with an average of 150 VMs per software component. In this case, the network had 50 data centers. Both figures show that the request solving time is approximately linear over the network and application sizes and for these cases it was always in the order of milliseconds. That behaviour is highly desirable to execute the meta-scheduler in a real time and interactive system.

4.7 Conclusion

Cloud computing is in continuous expansion and the number of servers will likely increase by a factor of 10 between 2013 and 2020 [10]. New infrastructure providers emerge to offer their computing capacity, and service providers develop new applications such as social networks, video on-demand, and big data analysis. In this context, a cloud broker organizes the multiplicity of resources and handles interoperability issues in a unique platform to deploy and monitor VMs across multiple data centers.

Even if commercial cloud brokers already exist, they lack a meta-scheduler that chooses the optimal data center for each application component. In this context, we formalized a dynamic meta-scheduler through a MILP model and developed a powerful heuristic that scales to hundreds of data centers and thousands of applications.

When compared to unrestricted cases in which only the cheapest data center is chosen for every application, the results of our meta-scheduler show 6 times shorter communication delay, 30% power savings, 60 times less CO_2 emissions, and optimal cost. Final users obtain a better QoS thanks to the reduction in communication delay; service providers optimize the cost paid to execute VMs and information transfer; infrastructure providers minimize power consumption, and the environment is conserved by an important reduction in the greenhouse gases. The meta-scheduler execution time is kept in the order of milliseconds for realistic case sizes, making possible the integration with the interactive system of a cloud broker.

CHAPTER 5

ONLINE, ELASTIC, AND COMMUNICATION-AWARE VIRTUAL MACHINE PLACEMENT WITHIN A DATA CENTER

Previous chapters of this thesis have presented the planning of cloud computing networks and the problem of placing applications across multiple data centers to reduce delay between VMs and users, improve the quality of service (QoS), and reduce cost, power consumption, and CO_2 emissions. This chapter focus on the daily management of a cloud data center that receives new applications that have a workload that changes over time. Section 5.1 explains the proposed strategy to assign VMs to servers. Section 5.2 describes the MIP model that optimizes QoS and power consumption. Section 5.3 proposes a tabu search based heuristic to solve the VM placement. Section 5.4 describes a case study of a data center with 128,000 servers including network topology, application topology, workload, and power consumption parameters. Section 5.5 presents the results obtained by comparing the proposed scheduler with static VM placement and dynamic VM placement with first-fit policy. Finally, the chapter is concluded in Section 5.7.

5.1 Virtual machine placement strategy

The algorithm proposed in this chapter is characterized as online because application requests are received consecutively, each one at a different time. For each application request, the decision to make is which server will host each requested VM for that application. After the decision is made, the hypervisors of the target servers create the VMs in real time. The objective is to minimize delay among VMs, maximize the network throughput available among VMs, and minimize the server energy consumption. The application elasticity is also taken into account, that is, the workload is expected to increase and decrease over time. Our placement strategy can be characterized as:

1. based on application graph,
2. communication-aware,
3. energy-efficient,
4. and elastic.

In what follows, we explain each of these points.

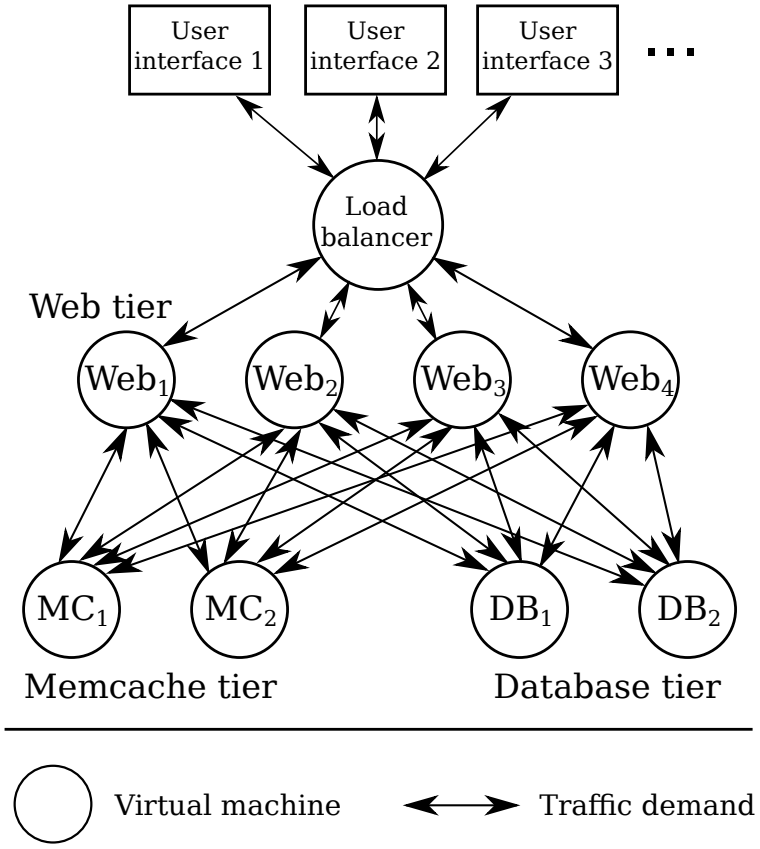

Figure 5.1 Graph of a multi-tier application.

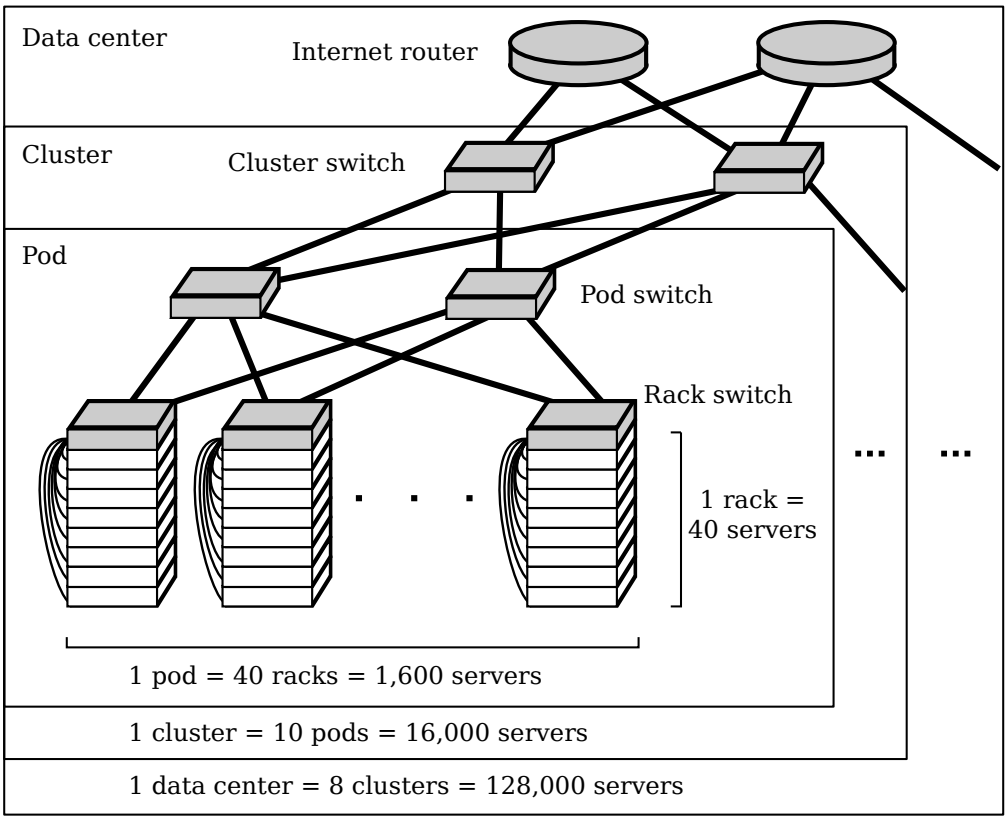

Figure 5.2 Data center network topology.

5.1.1 Application graph

The requirements of an application are modeled as an application graph where each node is a VM and each arc indicates a VM sending traffic to another one. The application developer defines the type of VMs needed and the expected traffic between each pair of VMs. The scheduler takes that information as an input to place the VMs. In this approach, the user needs to enter more information than in the current systems where only the types of VMs are defined. That additional information is used by the scheduler to provide more throughput among VMs, so the user would be interested in providing that information. Figure 5.1 shows an example of a multi-tier application graph, an architecture extensively used for the state of the art applications [50]. Each tier is composed of multiple VMs running in parallel to handle requests in an efficient way. VMs from the web tier receive user requests, query, process, and present the information. VMs from the memcache tier store objects in RAM to provide efficient access to information requested by the web tier [88, 50]. Different VMs store different objects, then a web VM can query multiple memcache VMs depending on the objects needed. This produces a many-to-many communication pattern between the web tier and the memcache tier. When an object is not present in a memcache VM, then it must be retrieved from a database VM. The third tier is composed of database VMs that read and write information on the permanent storage devices. At this level, the information is also split and different databases store different objects. Each web VM can query multiple databases, and each database can receive requests from multiple web VMs, producing a many-to-many communication pattern between web and database tiers. Given that the application's VMs communicate among themselves, the application performance depends on which server of the network will host each VM.

5.1.2 Communication

We state that the proposed model is communication-aware because the distance between servers in the network topology is considered. The model includes the delay among VMs in the objective function, thus VMs that communicate among themselves will be placed in nearby servers. Placing VMs that communicate among themselves in the same server is better than placing VMs in servers in the same rack, which in turn is better than servers in different racks. This also reduces the load on the links between switches, thus improving quality of service and avoiding bottlenecks to increase the throughput available among VMs.

The network topology used for the test cases in this chapter hierarchically connects 128,000 servers, as shown in Figure 5.2. A server rack is composed of 40 servers connected to a Top of Rack (ToR) switch through 1 Gigabit Ethernet. 40 server racks form a pod of 1,600

servers. The racks in a pod are connected to two pod switches through 10 Gigabit Ethernet. A cluster is composed of 10 pods, with two cluster switches and 16,000 servers. The pod switches are connected to the two cluster switches through aggregation of 10 Gigabit Ethernet links. The data center is composed of 8 clusters, and the cluster switches are connected to the Internet routers.

The following example illustrates the importance of taking into account the communication among VMs. Suppose that 20 servers of the first rack of Figure 5.2 are allocated to the VMs of an application a_1 that sends information to the Internet, and there are 20 available servers in the same rack. What would happen if a new application a_2 with 40 large VMs that communicate among themselves must be added to the network? A standard policy such as first-fit would ignore the communication among VMs and would place 20 VMs to fill rack 1 and 20 VMs in rack 2. In that case, the 10 Gbps links that connect the ToR switch of rack 1 to the pod switches will be used by the traffic of applications a_1 and a_2. On the other hand, a communication-oriented VM placement will place the 40 VMs in rack 2 and the traffic of a_2 will remain within the rack and will not use the uplinks. That way there will be less probability of congestion and the available throughput among VMs is higher.

5.1.3 Energy efficiency

The proposed approach is also energy-efficient because an active server with enough capacity will be preferred rather than using an inactive server. Furthermore, between two servers of different models or manufacturers, the one that consumes less power will be used. Inactive servers without VMs are in the suspend-to-ram ACPI state S3, that is the standard mode where the CPU and other devices are suspended, and the RAM remains powered to keep operating system information available for fast switching to the normal state. A server in suspend-to-ram state consumes less than 2% of the peak server power [89, 90, 72]. The transition time between suspend-to-ram and active states takes a few seconds allowing the switching to be done in real time when the first VM is placed on the server. When the server becomes active, the hypervisor starts the new VM, and the VM's operating system boots.

5.1.4 Elasticity

The application elasticity is also considered for the placement policy. Depending on the hour and the day, applications receive different workload sizes. Adding and removing VMs to existing applications can handle a varying workload. The proposed approach keeps a fraction of each rack empty to be used by VMs added in peak periods, and this way these applications benefit of a reduced delay and high throughput. Considering the example of

Figure 5.1 as the application in a particular period, then new web VMs can be added in the peak period to handle a larger workload. A placement policy that does not consider the application elasticity would fill all the servers in a rack with multiple applications in the normal period. Then in the peak period, the new web VMs would be placed in other racks that may be separated by multiple links from the original rack. On the other hand, the proposed elasticity aware policy will use the reserved servers in the same rack, increasing the QoS of traffic between new web VMs and existing memcache and database VMs.

Note that VM migration is not considered in the proposed approach because even if live migration can be done in a short time interval [58], that period could negatively impact the programs and network protocols executing on the VM. Because the algorithm can be implemented by an infrastructure provider with different types of customers, the assumption is that the provider will always want to increase the system availability, given that the loss in revenues could be much higher than the power savings because of VM migration. In fact, the elasticity awareness of our method that reserves space for VMs of the peak period prevents the fragmentation of applications and the need for VM migration. That being said, the proposed scheduling algorithm could be used to migrate VMs for cases that really need it. In an analogy with file systems, the most efficient ones separate files across the disk allowing each file to grow and shrink in contiguous sectors preventing the need of a defragmentation process [91]. In our case, we place applications in different racks to keep space available and avoid VM migration.

5.2 Problem description

5.2.1 Network topology

A cloud data center network is represented as a graph $G^N(V^N, A^N)$, as shown in Figure 5.3. The set of nodes V^N contains the servers $\mathcal{S}$, switches $\mathcal{Z}$, and a special node χ that represents destinations outside the data center network. For instance, χ can represent the users' computers that get information from the VMs. The data center contains a set of racks $\mathcal{R} = \{1, \ldots, |\mathcal{R}|\}$, and each rack $r \in \mathcal{R}$ accommodates a set of servers $\mathcal{S}_r \subseteq S$. The servers of a rack are connected to a ToR switch, and all the switches are connected through a particular topology. Each link direction is an arc in A^N.

5.2.2 Applications and scheduler

A program called *scheduler* receives requests to add, remove, and resize applications. Let a_n represent a particular application and $G^{a_n}(V^{a_n}, A^{a_n})$ the topology of that application. As can be seen in Figure 5.3, each node in V^{a_n} is a Virtual Machine (VM), and each arc

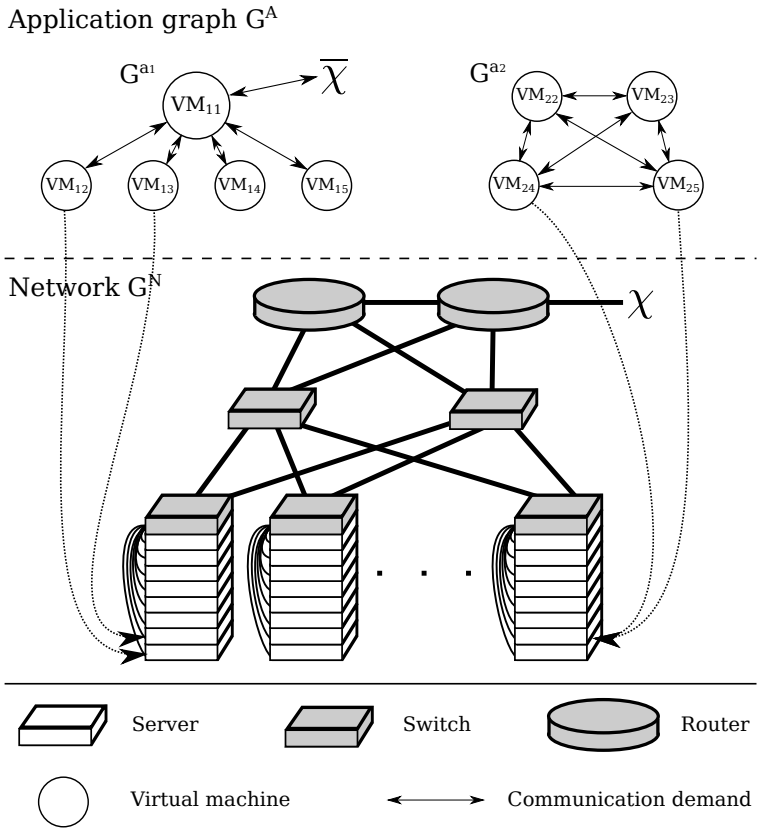

Figure 5.3 Application and network mapping.

$(i, j) \in A^{a_n}$ represents that VM i sends information to VM j. Similarly to the network layer, there is a special node $\overline{\chi}$ in V^{a_n} that represent programs outside the data center that exchange information with the VMs.

$G_t^A(V_t^A, A_t^A)$ is the graph that contains all the applications being executed at time t. The nodes in V_t^A are the VMs of all applications, and the arcs in A_t^A are the communication demands. If the system receives a request to add a new application a_n at time $t+1$ then:

$$G_{t+1}^A = (V_t^A \cup V^{a_n}, A_t^A \cup A^{a_n})$$

When a request to remove an application a_n is received at time $t+1$ then:

$$G_{t+1}^A = (V_t^A - V^{a_n}, A_t^A - A^{a_n})$$

M_t: $V_t^A \rightarrow \mathcal{S}$ is a mapping that specifies which server is assigned to each VM at time t. The rest of this section defines the parameters, objective function, and constraints of the MIP model to place the VMs of a new application a_n at time t. Then the scheduler has to define the mapping M_{t+1} starting from M_t.

5.2.3 Communication traffic

The VMs of an application can communicate among themselves and with Internet hosts to achieve the application's purpose. These communication demands are represented as arcs in the application graph. When the scheduler decides the VM placement and deploys VMs in servers, the communication demands will be routed through network paths. The proposed scheduler takes the traffic demands into account to calculate the communication delay among VMs and place VMs in nearby servers.

The parameters related to the communication traffic are the following:

b_d Average throughput of demand $d \in A_t^A$ (in bps).

$\overline{c_e}$ Capacity of link $e \in A^N$ (in bps).

ζ Average packet size (in bits).

5.2.4 Servers

Data centers typically have different server models. Each server model has capacities associated to each server resource:

$\mathcal{M}$ Set of server models, e.g., 1 = Dell PowerEdge R420, 2 = HP ProLiant Dl360, 3 = IBM System x3750.

μ_k Model of server $k \in \mathcal{S}$.

$\mathcal{K}$ Set of resources that each server has, which include 1 = CPU, 2 = network card, 3 = RAM, and 4 = storage.

c_{pm} Capacity of resource $p \in \mathcal{K}$ on a server of model $m \in \mathcal{M}$ in the corresponding units (GHz, Gbps, GB).

5.2.5 VMs

Different VM types require more or less of each server resource. Each application defines the type of VMs needed depending on the VM purpose. For instance, the cloud service Amazon EC2 defines an EC2 Compute Unit as the capacity of a 1 GHz 2007 Xeon processor. A small VM instance has 1 EC2 Compute Unit and 1.7 GB RAM, a medium VM instance has 2 EC2 Compute Units and 3.75 GB RAM, and a large VM instance has 4 EC2 Compute Units and 7.5 GB RAM [39]. There can also be VM types with a high amount of one of the resources. For instance, a web server would use a VM with high CPU allocation to process a high number of requests per unity of time. A memcache VM would be a VM with a high amount of RAM to increase the number of objects that can be stored in memory.

The following are the parameters that define the VMs:

$\mathcal{T}$ Set of VM types, e.g., 1 = small instance, 2 = medium instance, 3 = large instance, 4 = high RAM instance, 5 = high CPU instance.

ψ_i Type of VM $i \in V_t^A$.

r_{pv} Requirement of resource $p \in \mathcal{K}$ for a VM of type $v \in \mathcal{T}$ in the corresponding units (GHz, Gbps, GB).

$\mathcal{P}$ Set of possible assignments (v, m) indicating that a VM of type $v \in \mathcal{T}$ can be hosted on a server of model $m \in \mathcal{M}$.

5.2.6 Power consumption

The power consumption of a server is related to its resource utilization. Each server model has a different power consumption function. That function can be approximated through regression techniques based on measurements with different resource consumption levels. Then depending on the type of VMs to place in a server, the expected resource consumption can be calculated, and the expected power consumption can be estimated using the power function.

The proposed model considers that a server that does not host any VM can be put in suspended-to-ram state, and thus consume a very low amount of power. Because VMs can be added and removed dynamically we add a parameter to avoid activating and deactivating a server too often. If a server is turned on to host a VM, and then the VM is removed in a short interval of time, the server is kept on for the next VM to be added. Without that condition, the second VM could arrive while the server was entering in suspension state and would wait longer to start.

The following parameters define the power consumption aspect:

$w_m : \mathbb{R}^{|\mathcal{K}|} \rightarrow \mathbb{R}$
: Power consumed by a server of model $m \in \mathcal{M}$ as a function of CPU, network bandwidth, RAM, and disk allocated.

π_k Time of activation of server $k \in \mathcal{S}$ (in minutes); this value is -1 if the server was never activated.

h Minimal time interval that a server must be kept in active state (in minutes).

5.2.7 Elasticity

In order to group VMs of the same application together, we keep a fraction of each rack available for future VMs. That is defined by an expected utilization threshold on the rack resources (sum of the server resources). When the resources of a rack are higher than the threshold, a penalty term is added to the objective function.

u_{rpt} Utilization threshold of resource $p \in \mathcal{K}$ in rack $r \in \mathcal{R}$ ($0 \leq u_{rpt} \leq 1$).

5.2.8 System variables

The main variables for the scheduler are those that define in which server to place each VM. There are also decision variables that define whether a server is activated or not because that changes the system power consumption. Another variable is the traffic that flows in each link that is used to calculate the average transmission delay. Finally, we also need to track the resource consumption for each server and rack, and how much overflowed are the rack resources.

x_{ikt} 1 if VM $i \in V_t^A$ is located in server $k \in \mathcal{S}$; 0 otherwise.

y_{kt} 1 if server $k \in \mathcal{S}$ is in active state; 0 if the server is in suspend-to-ram state.

f_{det} Amount of traffic demand $d \in A_t^A$ carried by link $e \in A^N$ (in bps).

q_{kpt} Consumption of resource $p \in \mathcal{K}$ in server $k \in \mathcal{S}$ in the corresponding units.

l_{rpt} Consumption of resource $p \in \mathcal{K}$ in rack $r \in \mathcal{R}$ ($0 \leq l_{rpt} \leq 1$).

o_{rpt} Overflow of resource $p \in \mathcal{K}$ in rack $r \in \mathcal{R}$ ($0 \leq o_{rpt} \leq 1$).

5.2.9 Objective function

Three aspects are minimized in the multi-criteria objective function of the proposed approach: the average delay among VMs, the server power consumption, and an elasticity penalty. The first part of the objective to minimize is the communication delay among VMs. The transmission delay of each link is proportional to the packet size ζ and inversely proportional to the link capacity $\overline{c_e}$. This delay is used as a measure of the distance between servers in the network. Each link delay is weighted with the amount of traffic that it carries. The result is that the optimal placement will have VMs that communicate among themselves in nearby servers. The advantage of this delay formula when compared with the simple hop count is that it differentiates links with low and high capacity—e.g., 1 Gbps and 10 Gbps links.

$$C^D = \frac{\sum_{e \in A^N} (\zeta / \overline{c_e}) \sum_{d \in A_t^A} f_{det}}{\sum_{d \in A_t^A} b_d}$$

The second part of the objective to minimize is the server power consumption, that is calculated as the sum of each server power consumption:

$$C^E = \sum_{k \in \mathcal{S}} w_{\mu_k} \left(q_{k,1}, q_{k,2}, q_{k,3}, q_{k,4} \right)$$

The third term in the objective is an elasticity penalty that aims at leaving unused servers for new VMs created at the peak times. It is defined as the sum of the overflow of each rack, which starts to be positive when the allocated resources of a rack are above a threshold.

$$C^L = \sum_{r \in \mathcal{R}} \sum_{p \in \mathcal{K}} o_{rpt}$$

The objective function to minimize is the weighted sum:

$$z = \alpha C^D + \beta C^E + \gamma C^L \tag{5.1}$$

In this problem, the delay and power terms are typically non conflicting objectives because placing VMs in the same server reduces both delay and power consumption. Thus, minimizing both objectives is necessary and compatible. We choose to make the minimization of delay

as the first priority, the power consumption as the second priority, and elasticity as the third priority because that is the way to accomplish the proposed strategy. We illustrate that with the following case. When there is a server that is at half capacity in rack 1 and a new application with multiple VMs arrives, the scheduler can choose between placing a VM in the half used server or place it in another rack. Our strategy minimizes delay, so if the whole application fits in the first rack, then the server of that rack will be used. Otherwise, the application will be placed in a rack with enough capacity for the whole application. If the priorities were in the other way, that is more priority to power than delay, then the solution would essentially become a first-fit algorithm. In that case, whenever a server is half used, the next application will use that server and that rack. The scheduler would not have the freedom to choose a different rack to reduce delay and increase the throughput among VMs. The elasticity penalty is counted as the third priority because it reduces the delay of the VMs expected to come in the peak time by leaving rack resources available for them. Increasing the priority of the elasticity term would mean to allow more delay in the present to reduce delay of the potential future VMs. In that case, the overall delay would not be reduced. For these reasons, we choose the order of priorities 1) delay, 2) power consumption, and 3) elasticity.

With that strategy, the coefficients must be chosen to make the terms of the objective function $\alpha C^D > \beta C^E > \gamma C^L$ for every feasible solution. That way, between two solutions, the delay term will first determine which has the lowest value, then power consumption, and then elasticity. In an implementation, a large integer type is used for the objective value, and the integer's digits are split for each term. The number of digits of each term are calculated between the lowest and highest values that the term can take. Then, the coefficient of a term is the power of 10 that leaves enough zeros for the lowest terms. In our cases a 128 bits integer was used to hold the three terms.

5.2.10 Constraints

- *VM placement*

$$\sum_{\substack{k \in \mathcal{S} / \\ (\psi_i, \mu_k) \in \mathcal{P}}} x_{ikt} = 1 \qquad \forall \quad i \in V^{a_n} \tag{5.2}$$

$$\sum_{\substack{k \in \mathcal{S} / \\ (\psi_i, \mu_k) \notin \mathcal{P}}} x_{ikt} = 0 \qquad \forall \quad i \in V^{a_n} \tag{5.3}$$

$$x_{ikt} = 1 \qquad \forall \quad (i,k) \in M_{t-1} \tag{5.4}$$

$$x_{ikt} = 0 \qquad \forall \quad i \in V_{t-1}^{A}, k \in \mathcal{S}, (i,k) \notin M_{t-1} \tag{5.5}$$

Equations (5.2) and (5.3) define that each VM must be placed in one server with a model that is feasible for that type of VM. Equations (5.4) and (5.5) state that the VMs previously placed remain in the same servers.

- *Active servers*

$$y_{kt} \geq x_{ikt} \qquad \forall \quad i \in V_t^A, k \in \mathcal{S} \tag{5.6}$$

$$y_{kt} = 1 \qquad \forall \quad k \in \mathcal{S}/t \in [\pi_k, \pi_k + h] \tag{5.7}$$

Constraint (5.6) states that the servers hosting VMs must be in active state. Equation (5.7) says that servers must remain active at least h minutes.

- *Flow conservation*

$$\sum_{e \in \Gamma_k^-} f_{det} + x_{ikt} b_d = \sum_{e \in \Gamma_k^+} f_{det} + x_{jk} b_d \tag{5.8}$$
$$\forall \quad d = (i,j) \in A_t^A, \quad k \in \mathcal{S}$$

$$\sum_{e \in \Gamma_k^-} f_{det} = \sum_{e \in \Gamma_k^+} f_{det} \tag{5.9}$$
$$\forall \quad d = (i,j) \in A_t^A, \quad k \in \mathcal{Z}$$

$$\sum_{e \in \Gamma_\chi^-} f_{det} = b_d \qquad \forall \quad d = (\overline{\chi}, j) \in A_t^A \tag{5.10}$$

$$\sum_{e \in \Gamma_\chi^+} f_{det} = b_d \qquad \forall \quad d = (i, \overline{\chi}) \in A_t^A \tag{5.11}$$

Equation (5.8) relates VM placement variables and the routing of the traffic demands. It associates the application and the network layers. For a demand from a VM $i \in V_t^A$ placed in server $k_1 \in \mathcal{S}$ to a VM $j \in V_t^A$ placed in server $k_2 \in \mathcal{S}$, the flow of traffic b_d starts in k_1 and ends in k_2. Equation (5.9) states that in each switch k of the path from k_1 to k_2 the flow is conserved, that is the incoming flow of k is equal to its outgoing flow. Equation (5.10) states that the traffic coming from outside the data center enters by node χ. Equation (5.11) states that the traffic going outside the data center exits by node χ.

- *Link capacity*

$$\sum_{d \in A_t^A} f_{det} \leq \overline{c_e} \qquad \forall \quad e \in A^N \tag{5.12}$$

In Constraint (5.12), the flow of each link is lower than its capacity.

- *Server resource utilization*

$$q_{kpt} = \sum_{i \in V^A} r_{p\psi_i} x_{ikt} \qquad \forall \quad k \in \mathcal{S}, p \in \mathcal{K} \tag{5.13}$$

$$q_{kpt} \leq c_{p\mu_k} \qquad \forall \quad k \in \mathcal{S}, p \in \mathcal{K} \tag{5.14}$$

Equation (5.13) calculates the amount of resource p reserved in server k. Constraint (5.14) requires that the capacity of each resource is respected.

- *Rack resource utilization*

$$l_{rpt} = \frac{\sum_{k \in \mathcal{S}_r} q_{kpt}}{\sum_{k \in \mathcal{S}_r} c_{p\mu_k}} \qquad \forall \quad r \in \mathcal{R}, p \in \mathcal{K} \tag{5.15}$$

$$o_{rpt} \geq \frac{l_{rpt} - u_{rpt}}{1 - u_{rpt}} \qquad \forall \quad r \in \mathcal{R}, p \in \mathcal{K} \tag{5.16}$$

$$o_{rpt} \geq 0 \qquad \forall \quad r \in \mathcal{R}, p \in \mathcal{K} \tag{5.17}$$

Equation (5.15) aggregates the server resource consumption for a whole rack and normalizes it between 0 and 1. Constraints (5.16) and (5.17) defines the overflow variable for each rack and resource. That variable starts to be positive when the rack load l_{rpt} is above the utilization threshold u_{rpt} and reaches the value of 1 when the rack load is 1.

- *Domain of variables*

$$x_{ikt} \in \{0, 1\} \qquad \forall \quad i \in V_t^A, k \in \mathcal{S} \tag{5.18}$$

$$y_{kt} \in \{0, 1\} \qquad \forall \quad k \in \mathcal{S} \tag{5.19}$$

$$f_{det} \in \mathbb{R}_{\geq 0} \qquad \forall \quad d \in A_t^A, e \in A^N \tag{5.20}$$

$$q_{kpt} \in \mathbb{R}_{\geq 0} \qquad \forall \quad k \in \mathcal{S}, p \in \mathcal{K} \tag{5.21}$$

$$l_{rpt} \in \mathbb{R}_{\geq 0} \qquad \forall \quad r \in \mathcal{R}, p \in \mathcal{K} \tag{5.22}$$

$$o_{rpt} \in \mathbb{R}_{\geq 0} \qquad \forall \quad r \in \mathcal{R}, p \in \mathcal{K} \tag{5.23}$$

Finally, we define the domain of each variable. They are all non-negative. The placement

and server switching variables are binary, and the flow, server utilization, rack utilization, and rack overflow variables are real numbers.

5.3 Solution approach

Given that a VM scheduler in a cloud data center could need to scale to more than 100,000 servers and VMs, we developed a very efficient hierarchical heuristic based on tabu search. We name the proposed method Cloptimus Scheduler (CS). We exploit the fact that data center topologies are split in clusters and sub clusters (pods). As the topology described in Section 5.1.2, each pod in the test cases contains 1,600 servers that can host a total of 6,400 VMs. Then each pod has a separate scheduler that handles VMs assigned to it. Each pod is split in racks of servers—e.g., 40 racks of 40 servers—. Each server in a rack is symmetrical to the other servers in the same rack because they are all connected to the same rack switch with the same link capacity.

The general structure of the heuristic shown in Algorithm 3 is similar to the problems of Chapters 3 and 4 but has important differences in the neighborhood relation between solutions, objective function calculation, tabu list size, number of iterations before stopping the execution, and in the possibility to resize applications on the fly.

In this problem, a solution S is a mapping between VMs and servers. The neighborhood $\mathcal{N}(S)$ is the set of solutions that differ in the placement of one of the VMs that is being scheduled. To take advantage of the symmetry between solutions, each possible movement is to move each VM to the first available server in each rack. Thus, the number of possible movements is reduced to the number of VMs to schedule multiplied by the number of racks in a pod. The reduction of movements is valid because the servers in the same rack are equal in terms of delay. Furthermore, choosing the first one available optimizes the second objective that is the power consumption because a server will be activated only if the previous ones were full. This assumes that servers are sorted by energy efficiency in each rack. Thus, the first available server will also be the most efficient one among all availabilities.

The number of iterations q to keep a movement in the tabu list is the square root of the neighborhood size. In this case, a second restriction was added to the tabu movements. Each VM that is moved to a server is then kept in that server for a number of iterations, that is half of the number of VMs to schedule. Keeping a VM fixed in a new rack for a number of iterations allows the other VMs to be moved to that rack, and that solution with the whole application in a different rack can be evaluated by the algorithm. When the solution does not improve for *MAX_ITERATIONS* the algorithm stops. That value was set to the number of VMs to schedule multiplied by the number of racks.

Algorithm 3 Add, remove, or resize an application.

```
function CloptimusScheduler(Request, M)
    a_n ← Request application
    if Request type is add then
        M ← InitialGreedySolution(a_n, M)
        M ← TabuSearch(a_n, M)
    end if
    if Request type is remove then
        Remove a_n's VMs from M
    end if
    if Request type is resize then
        Remove a_n's VMs from solution M
        Add or remove VMs from a_n
        Place existing VMs in the same servers than before in M
        if a_n has new VMs to schedule then
            M ← InitialGreedySolution(a_n, M)
            M ← TabuSearch(a_n, M)
        end if
    end if
end function

function InitialGreedySolution(a_n, S)
    for each vm in vmsToSchedule(a_n) do
        Move vm to the server that least increases z(S)
    end for
    return S
end function

function TabuSearch(a_n, Current)
    S ← Current
    Best ← Current
    L ← {}
    i ← 0
    repeat
        Choose the pair (c, s) ∈ N(S) − L that minimizes z when moving vm to s in S.
        Move c to s in S
        Add (c, s) to tabu list L for q iterations.
        i ← i + 1
        if z(S) < z(Best) then
            Best ← S
            i ← 0
        end if
    until i = MAX_ITERATIONS
    return Best
end function
```

5.4 Case study

5.4.1 Network topology, delay, and throughput

The data center topology connects 128,000 servers, split in pods of 1,600 servers, that are in turn split in 40 racks of 40 servers as shown in Figure 5.2 and described in Section 5.1.2. The proposed scheduler is executed as multiple parallel processes, each one handling a pod of 1,600 servers. The experiments shown in this chapter uses one scheduler managing the first pod of the topology.

The delay of each link was set as the time to transmit one packet of 1,500 bytes at the line capacity: 12 µs at 1 Gbps and 1.2 µs at 10 Gbps. That delay is used as a distance between servers. The servers in a rack are connected through the rack switch, and thus they are at a distance of 2 links Gigabit Ethernet, or 24 µs. Servers in different racks but in the same pod have a distance of 2 links Gigabit Ethernet and 2 links 10 Gigabit Ethernet, or 26.4 µs.

Even if there is only 2.4 µs in the transmission delay when placing two servers in the same rack instead of two servers in different racks, minimizing delay also implies increasing the available throughput between servers in a great measure. If the 40 servers of a rack send information to servers in the same rack, they could achieve an aggregated throughput limit of 40 Gbps. On the other hand, if the 40 servers send information to other rack, the limit would be 20 Gbps, that is the sum of the two uplinks. Keeping the traffic within the rack decreases the utilization of inter-switch links, thus the probability of congestion, and the queuing delay.

5.4.2 Application topology

Each application to be scheduled is a multi-tier application with a different number of VMs. Each application is composed of 1 load balancer, 1 memcache, 2 databases, and a number of web servers that changes depending on the workload size. That is the same architecture as that of Figure 5.1. Each web server VM uploads up to 100 Mbps of traffic to its load balancer. The load balancer forwards the traffic of its web VMs to the Internet. 50% of the traffic uploaded by the web server is retrieved from the database, and the other 50% is retrieved from memcache.

The number of web VMs of each application was drawn from a Pareto distribution with parameters $\alpha = 1.6$ and $\beta = 1$. Generating 800 applications gave between 1 and 68 web VMs per application.

The workload variation is based on the load of a whole day in an Internet link of a data center in San Jose, California, publicly available by the CAIDA initiative [4]. As shown in Figure 5.4, that link has periods of high utilization (3.72 Gbps) and low utilization (3.22

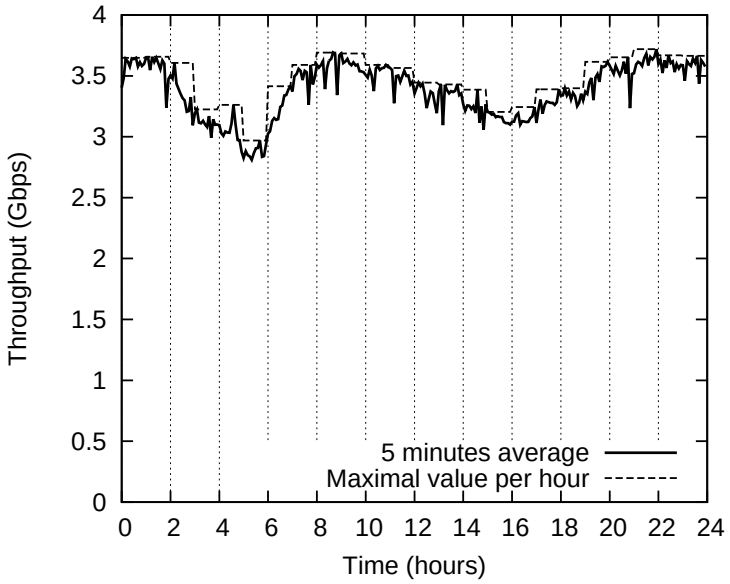

Figure 5.4 San Jose CAIDA's traffic monitor during August 3, 2013 [4].

Gbps). The relative variation in each period was used to multiply the number of web VMs of each application each hour of the day.

5.4.3 Servers

In these cases, each server has 8 cores at a frequency of 2 Ghz and 64 GB of RAM. Each core is shared by two virtual CPUs. Each VM requires 4 virtual CPUs of 1 GHz and 16 GB RAM, the same requirements as the extra large instances of Amazon EC2 [39]. With these settings, each server can host up to 4 VMs.

5.4.4 Power consumption

We considered two server models with the same specifications but different power consumption. One server model is energy-efficient and consumes 10 W in suspended state, 110 W in idle state, and an average of 210 W when it hosts 4 VMs. The other server model is less efficient and consumes 20 W in suspended state, 150 W in idle state, and an average of 350 W when it hosts 4 VMs [89, 90, 72]. That average power consumption can be seen in Figure 5.5 as a function of the number of VMs in a server.

Each rack has either all energy-efficient servers, or all energy-inefficient servers. One of the cases tested had all racks with energy-efficient servers. Another case had energy-inefficient servers in racks numbered with odd numbers, and energy-efficient servers in racks numbered with even numbers.

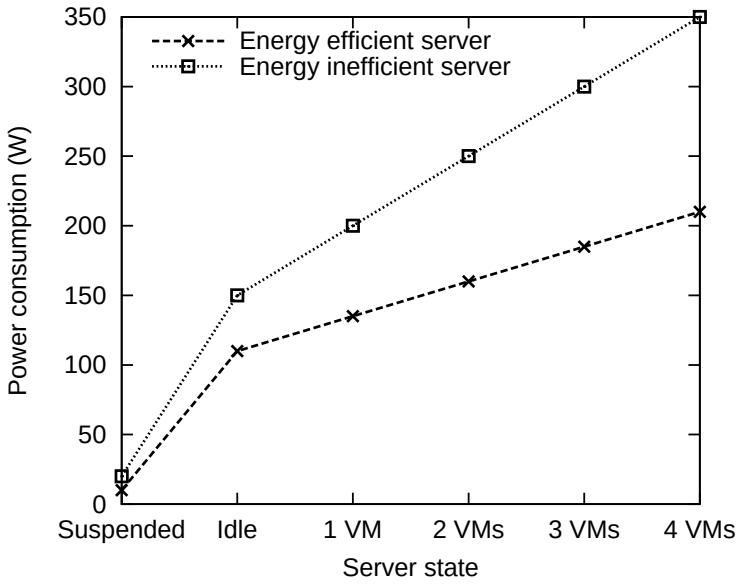

Figure 5.5 Server power consumption.

5.5 Results

This section analyzes the proposed Cloptimus Scheduler (CS) for the case study, and compares CS with First Fit (FF) policy. We first analyze an initial period when the applications are deployed, and then the whole day with a varying workload.

5.5.1 Initial period

We first added 800 applications with a total of 5,027 VMs corresponding to the workload between 5 PM and 6 PM to a pod of 1,600 servers. For each application, CS uses the tabu search algorithm to place its VMs. In another experience with the same applications and VMs, the policy to place each VM was FF, that is to choose for each VM the first server with enough capacity.

Table 5.1 shows the results obtained by the two policies. The average transmission delay was 70% higher in FF than in CS, and the average link utilization was 9.9% higher in FF than CS. That is because CS has the minimization of delay as first priority, and FF does not take traffic into account to place VMs. When an application with multiple VMs is deployed, FF picks the first the server for the first VM. If that server has not enough space for the second VM, then it will be placed in a second server. The traffic between these VMs will use the links between the servers and the rack switch. On the other hand, CS will pick a server

Table 5.1 Cloptimus Scheduler vs First-Fit after adding 800 applications.

Solution:	CS	FF
Avg. delay	11.3 µs	19.2 µs
Avg. link utilization	13.1%	23.0%
Avg. inter-switch link util.	10.5%	11.9%
Max. inter-switch link util.	32.2%	36.7%
Number of VMs	5,027	5,027
Number of servers	1,258	1,257
Total power	267.5 kW	267.4 kW

with enough space for two VMs, and the traffic between both VMs will be kept within the server and will not use any link.

Considering the links that connect rack and pod switches, the average link utilization was 1.4% higher in FF than CS. That improvement is because CS minimizes delay by placing the VMs of each application in a single rack, and reduces the use of inter-rack links. Because FF places the VMs of an application in consecutive servers, they will often be placed in the same rack. However, when a rack is almost full, an application will be split and the links between rack and pod switches will be used. As we will see, the difference between CS and FF is larger when applications are resized over the day.

With respect to power consumption, both mechanisms are power-efficient because they tend to fill an active server with VMs before turning on an inactive server. In some cases, CS will temporally consume more power because it will prefer to turn on a server to place VMs of the same application in the same rack. However, the VMs of the next applications will use the server that had been kept partially occupied, and the total number of servers used is almost the same.

CS minimizes power consumption by taking into account the consumption of each type of server. In the case shown, all the racks have servers with the same type of energy consumption. When racks of energy-efficient servers and energy-inefficient servers are alternated, CS consumes 332 kW and FF, 357 kW, an 8% advantage for CS.

5.5.2 Workload

Once the 800 applications are deployed, we consider a workload that varies over the day. Each application is resized each hour according to the maximal workload expected for that hour. New VMs are deployed when the workload increases in size, and VMs are removed in periods where the workload decreases. Figure 5.6 shows the number of VMs used to match

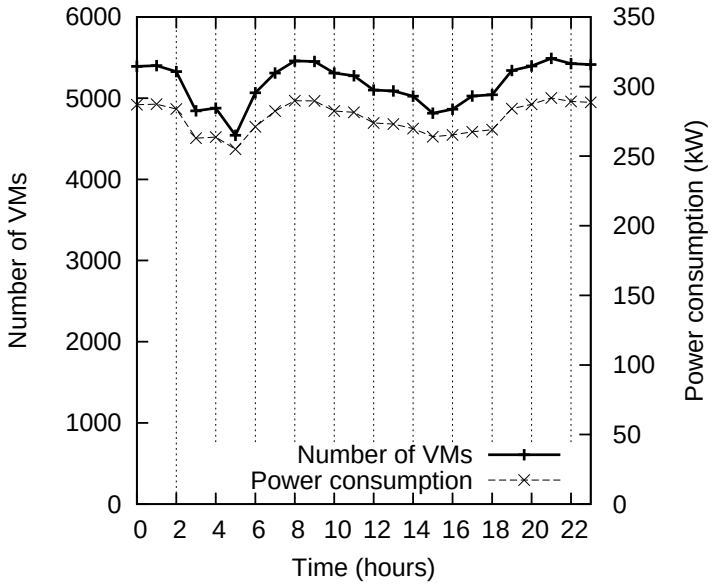

Figure 5.6 Number of VMs used and power consumption in each period.

the workload in each period and the power consumption achieved by CC in a pod of 1,600 servers.

First, we compare how much energy is saved by considering the variation of workload instead of dimensioning for the peak period. In this case, the peak period is at 9 PM when 291.8 kW are consumed by the VMs placed by CS in a pod, or 23.3 MW if the pod power consumption is extrapolated to the whole data center. If the servers used in that peak period consume that power during a year, 204.4 millions kWh would be consumed. Multiplying by an electricity price of $0.16 / kWh [92] gives a total of $32.7 millions for the electricity employed by the data center during a year. Using CS to resize applications during the day and suspending unused servers saves 4.9% of energy consumption, that becomes $1.6 millions per year. These savings are from the cloud provider point of view. Users that deploy applications would achieve higher savings in the cost paid to the provider to host VMs. Applications whose workload have more variability during the day benefit the most.

CS is then compared to the FF policy at the time of resizing applications each hour of the day. Table 5.2 presents results of both policies in the peak period. The first remark that can be highlighted is that FF produces 47% more delay than CS. That also implies that the links are 10.3% more used, thus increasing the queuing delay and degrading the QoS. In particular, links that connect rack switches with pod switches reach up to 80% utilization in FF. That is because 32 racks are used and 8 racks remain free after placing the applications in the initial period. VMs added in high activity periods are then placed in the last racks.

Table 5.2 Cloptimus Scheduler vs First-Fit in the highest period (9 PM).

Solution:	CS	FF
Avg. delay	13.5 µs	19.9 µs
Avg. link utilization	20.7%	31.0%
Avg. inter-switch link util.	15.2%	27.3%
Max. inter-switch link util.	47.0%	80.0%
Number of VMs	5,490	5,490
Number of servers	1,385	1,373
Total power	291.8 kW	290.6 kW

The traffic of the new VMs is directed to the load balancers placed in the first racks using the links between rack and pod switches. On the other hand, CS uses 40 racks from the starting period leaving available space in each rack for VMs that arrive in high periods. Then, CS places each new VM in the rack that hosts the corresponding application and thus reduces the link utilization between racks and improves the QoS. The cost to pay is that 12 more servers are used in the pod of 16,000 servers, and power consumption increases 0.4% compared to FF in the case where all servers are equally energy-efficient. The energy consumed in the whole day is 0.3% higher in CS than FF because a few more servers were used to improve the QoS.

5.5.3 Execution time

The execution time of CS was evaluated for different application sizes and number of servers. To vary the application size, an empty pod with 1,600 servers was gradually filled with applications with between 6 and 40 VMs each. For each application, the time to schedule the application was recorded. With that information, the average execution time for each application size was calculated as seen in Figure 5.7.

To analyze the variation of scheduling time as a function of the network size, the number of VMs in each application was set as 20 and we varied the number of racks, with 40 servers per rack. For each network size, applications are scheduled until all the servers are full with VMs. The average scheduling time was calculated for each test. Figure 5.8 shows that the scheduling time of an application grows linearly as a function of the number of racks or servers. In particular, scheduling an application with 20 VMs in a pod of 1,600 servers take an average of 283 ms.

Concerning the workload variation over the day, the scheduler must resize each application each hour of the day. That implies choosing the servers to place new VMs in the periods when an application expands. The fact that only the new VMs are deployed makes the

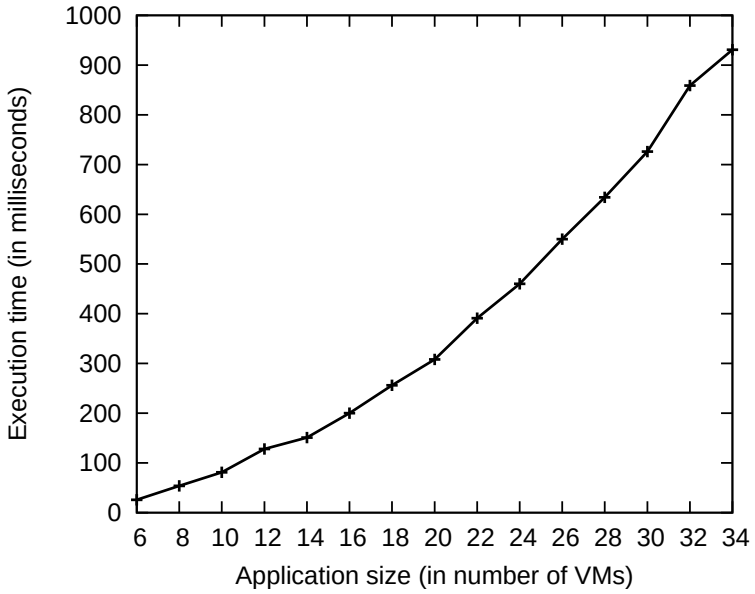

Figure 5.7 Average scheduler execution time over the application size.

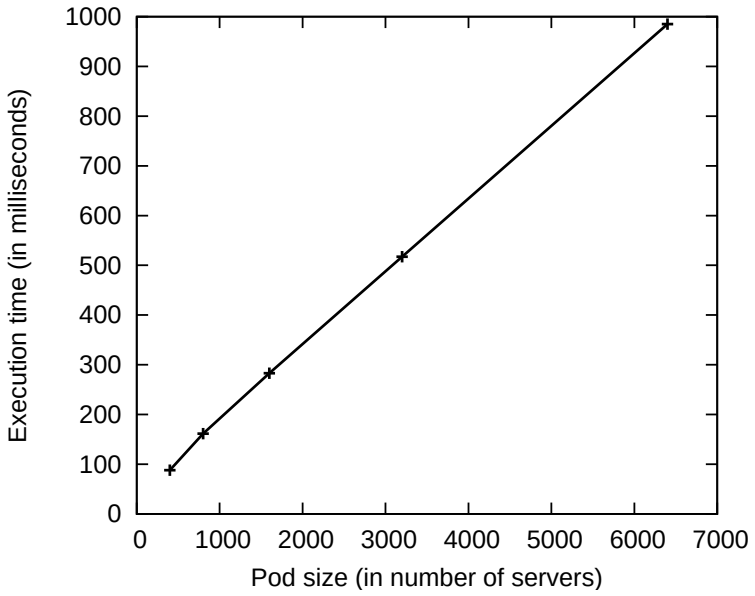

Figure 5.8 Average scheduler execution time over the network size.

operation very quick compared to the operation of initial application deployment. Adding 800 applications with their 5,027 VMs in a pod of 1,600 servers took 33 seconds in the initial period. Resizing those same applications only took a total of half a second in each period.

5.6 Discussion

The method proposed in this chapter shows the importance of taking the traffic among VMs into account to improve the QoS. The minimization of delay implies that VMs that communicate among themselves are placed in nearby servers and have more communication bandwidth available. This also provokes that links are less used and thus reduces the queuing delay. The reduction in delay, probability of congestion, and increase in throughput implies that the application QoS is improved.

An alternative to reduce the queuing delay and increase throughput among VMs is to augment the link capacity between switches. Topologies such as Fat tree [93] and VL2 [94] achieve full bisectional bandwidth and each pair of servers can communicate close to the maximal capacity of their network cards. However, that approach has a high cost in number of switches and power consumption that could be not justified when not all the applications require high throughput among VMs. These topologies are indeed useful for specific applications that do require an all to all communication pattern such as Hadoop clusters.

Experiences in this chapter study communication delay, throughput, and link utilization, metrics that vary depending on which server host each VM. The server processing time was not included because the processing power is a fixed requirement of each VM type, e.g., in number of virtual CPUs in a specific frequency. The number of VMs that will be assigned to an application will determine the number of requests per unity of time that each VM will handle and its processing time. However, the total response time is composed of the processing time of each VM that participates, e.g., load balancer, web server, database, memcache, and the communication delay. The way these times are combined is difficult to predict precisely because that will depend how the application is implemented. For instance, if a web VM makes requests to multiple databases in parallel or in a sequential fashion. We believe that the relation between communication delay and total response time should be analyzed through measurements in specific implementations. The fact that the transmission time is expressed in microseconds should not mislead to think that is negligible compared to a total response time in milliseconds because each application request can require multiple internal requests among VMs and because if congestion is not avoided the queuing delay could increase. As this chapter shows, our approach reduces congestion and communication delay and the total response time as a result.

5.7 Conclusion

This chapter presented an online method to place VMs in servers that optimizes communication among VMs and power consumption. The proposed scheduler considers server heterogeneity and VMs with different types of requirements. Taking advantage of regular network topologies makes possible to scale in the delay calculation and parallelization on multiple clusters. Furthermore, the proposed scheduler considers the elasticity of applications to improve the QoS of VMs that arrive in peak periods.

The method was formalized through a Mixed Integer Programming (MIP) model and a heuristic was developed scaling to more than 100,000 servers and applications. Applications can be added and removed on the fly, and the number of VMs of an application can be changed to match the workload that varies over the day. A case study shows the advantages of the proposed approach. Compared to first-fit policy, the delay among VMs is reduced by 70% and the most used network links are decreased a 33%. We also found that a 4.9% of power consumption is saved compared to statically provisioning of resources for the peak period. In a large data center with 128,000 servers, these annual savings represent $1.6 millions when the energy is charged $0.16 / kWh. Using first-fit in each period of the day taking into account the varying workload achieved 0.3% less energy consumption than our approach. That is because a few more servers were activated, which is a small price to pay that gives important benefits on the QoS. These results show that the proposed scheduler can be a key element in the management of a cloud provider contributing competitive advantages in QoS, energy consumption, and costs.

CHAPTER 6

GENERAL DISCUSSION

This thesis presented methods for planning cloud computing networks, and methods for their optimal management. Results showed that the proposed techniques can provide large benefits through the optimal location of data centers and applications as well as in the management of each data center.

Multiples actors can be interested in implementing these methods because the cloud is in full expansion. New infrastructure providers will build new data centers and they will need to define their locations. Current infrastructure providers will need to expand their networks to provide more resources to users. Furthermore, emerging countries are willing to build their own data centers to store the information of their citizens [95].

Regarding existing literature on data center location, we contributed with a flexible and multi-criteria model as well as a very efficient algorithm to plan any kind of networks, including the world largest ones. Service providers need to manage applications across multiple data centers in a practical and flexible fashion. That is where the proposed green cloud broker comes into action. On the one hand, cloud computing eases users to access applications. Installing many programs in each computer is no longer needed, the web browser of computers and cell phones can be used to access applications. On the other hand, organizations have increasingly more programs to develop, maintain, and allocate resources to. Multiple reasons—e.g., QoS, costs, system requirements, deployment history—cause the applications to be hosted in multiple data centers provoking challenging problems of interoperability and monitoring. In that context, a cloud broker becomes indispensable for an optimal resource management.

Cloud expansion as well as all human activities entail environmental concerns. On the one hand, video conferences, teleworking, and social networks could imply that people move less. That would be positive from an environmental point of view because moving bits is typically cleaner than moving persons. Also, the trend of smart mobile phones makes people use fewer desktop computers, thus reducing the power consumption since mobile devices are optimized to consume less battery. On the other hand, data centers and cellular antennas consume more and more energy. Therefore, the efforts of limiting power consumption as proposed in this thesis should be increased, and renewable energy should be rewarded [1].

Showing users how much they can improve the environment has a big impact on their habits. In fact, studies show that monetary penalties such as a carbon tax has less impact

than social pressure [96]. If a user that consumes a lot of energy is shown how much its neighbors consume, the user would be inclined to change habits. In the context of cloud management, this is achieved through the calculation and visibility of CO_2 emissions. The user could place applications in the cheapest data center, but we propose a tool that shows her/him how much less CO_2 would be emitted if other data centers are used, and how much that would cost. Furthermore, the tool can show organizations how many trees should plant to make their applications carbon neutral, given the fact that each tree can absorb 22 kg of CO_2 per year [97].

In the domain of coordinated resource management across multiple data centers, we believe to be the first to provide this environmental approach as a green cloud broker. We also contributed with a model that extends meta-scheduling architecture to a graph of software components that communicate among them. This is a very important aspect for the QoS of applications with geographically distributed users. The proposed algorithm has the efficiency that makes it suitable to be integrated in an interactive cloud broker.

6.1 Applications

The potential applications for each problem are here summarized.

- Chapter 3. The cloud planing problem is first useful from the research point of view to understand each aspect involved in the location of data centers and network planning. Such comprehensive model can be used by global organizations with their private networks such as Google, Facebook, and Akamai. These actors can make every decision of the whole problem.

- Chapter 4. The dynamic optimization problem can be used by a cloud broker that offers customers to choose among multiple data centers. Every type of application that is executed on a cloud can be customer of the cloud broker. Some examples are 1) multi-tier applications that are composed of web servers, databases, and load balancers, 2) global enterprise applications with offices located on multiple geographic points, 3) trading applications that execute on multiple countries.

  The cloud broker can be used for applications that use only one data center, and can also be used by applications distributed across multiple data centers. For applications in a single data center, a green cloud broker is useful to query a database of data centers taking into account multiple criteria. Multiple data center applications are more complex and they must be prepared to synchronize multiple databases. They also need global load balancing systems. These applications can most take advantage of the green cloud broker because the optimization is more complex to achieve.

- Chapter 5. The optimization of the VM placement can be used by every cloud data center to decide in which server place each VM. Multiple server types are considered in the model, and the algorithm scales to every data center size. The optimization engine must be plugged to a deployment module (custom or standard as OpenStack) that communicates to the hypervisor of each server in the data center. This system works in an autonomous fashion. A customer request VMs for a new application through a web page. That request is received by Cloptimus which has information on the current state of the network. Cloptimus optimizes and decide which server assign to each VM. The deployment system is used to communicate to each server chosen and start the VM requested. A server starts the VM and the operating system within the VM starts. Then, the user can connect to their VMs and install and use applications. Once the VMs are running, a load balancer can be used to spread requests across all the VMs.

6.2 Scalability

We here summarize what are the instance sizes used in this thesis.

- Chapter 3. Planning problem. Our test cases had 1,000 potential data center locations to make the algorithm scale to every possible case. Typically, organizations have few large data centers. Extreme cases with 1,000 small data centers can also be solved. Nygren et al. [12] show that Akamai has more than 1,000 sites with servers across the world. Akamai also allows users to execute VMs in those locations, what they call *edge computing*. Our approach allows a provider to evaluate the convenience of such distributed architecture and to select optimal locations for these points of presence. Similar to the cases shown in this thesis, there would be a trade-off between delay and power consumption, cost, and CO_2 emissions. Using fewer sites is cheaper, reduces power consumption and it is easier to operate, but introduces more delay. Our approach allows planners to evaluate these trade-offs.

- Chapter 4. Cloud broker. The test cases have up to 1,000 data centers. This can be used by a cloud broker that has information about data centers across the world.

- Chapter 5. The VM placement algorithm can be executed in parallel on multiple clusters. In our execution environment, each instance of the program can handle a pod of 1,600 servers in real-time. Any number of pods can be used. For instance, a large data center with 128,000 servers can be handled with 80 instances of the program.

6.3 Service architecture

Our approach considers distributed applications represented as a graph of software components that exchange information. For instance, multi-tier applications can be represented as a graph where each tier is a software component. Tiers of different regions can be different software components, and then the optimization decides which data center hosts each software component. This is for the broker optimization. Of course, such distributed application must be developed to execute on multiple data centers. Within a data center, an application is represented as a graph of virtual machines. Cloptimus places each VM in a server to optimize the quality of service and power consumption. Typically, each software component uses multiple VMs and a load balancer spread the load among them.

6.4 Elasticity

The VM placement approach described in Chapter 5 handles application elasticity. This allows providers to save power consumption and users to save money paid to providers. Each application has a different type of workload. Some of them present periodic fluctuations over time. Elasticity is the feature to dynamically resize an application to match the workload. As presented in Section 2.3 of the literature review, there are reactive and predictive methods to perform dynamic scaling. Cloptimus could be used with both approaches. A predictive model such as Moving Average or Exponential Smoothing can be used to calculate the number of requests per second in each hour of the day. Because each VM can process a number of requests per second with the expected quality of service, then the number of VMs in each period can be calculated. Our VM placement approach can be used to optimize the placement of these VMs during the day.

CHAPTER 7

CONCLUSION AND RECOMMENDATIONS

This thesis addressed the planning and management of cloud computing systems. These systems allow users to deploy applications accessible from any device in the Internet. Also, cloud computing reduces power consumption through server virtualization. The numbers of users and applications are continually expanding, and the cloud is integrated to everyday life of a big part of world population. That raises challenges on multiple aspects. Users require applications with high availability and quality of service (QoS). Providers need to minimize the costs of computing resources and power. Developers require tools that simplify the management of an increasing number of applications. Moreover, the cloud expansion entails a growth in the power consumption and in the CO_2 emissions to generate that power, to the point that if the cloud was a country, its power consumption would be the 5^{th} worldwide.

In this context, we proposed to address the issues of QoS, cost, power consumption, and pollution in three stages of planning and management: 1) planning a cloud computing network from the ground, 2) daily management of applications across multiple data centers, and 3) management of Virtual Machines (VMs) in a data center.

At the planning stage, one of the main decisions to make is where to locate data centers. That decision impacts the cost, QoS, power consumption, and CO_2 emissions. When data centers are close to users, the communication delay is shorter. The cost varies depending on the energy price and land. The power consumption is smaller when the surrounding air is used to reduce the consumption of air conditioning. CO_2 emissions are drastically reduced when data centers are provided with green energy. With respect to existing literature, we contributed with a new model that takes the distributed nature of cloud applications into account and a multi-criteria approach so planners are able to analyze multiple solutions regarding cost, QoS, power consumption, and CO_2 emissions. Furthermore, the assignment of link capacities and routing are simultaneously solved with the location problem. A tabu search heuristic was developed to solve cases with up to 1,000 potential data center locations in less than 10 minutes. Comparing with optimal solutions, the heuristic gives results with optimality gaps between 0% y 1.95%. Thus, planners can use the algorithm for any type of network and achieve important benefits regarding the stated objectives.

The second stage is the everyday management of distributed applications. In the same line of location problems, the optimization is about which data center will host each software component at each time because that impacts on the multiple objectives stated. The ap-

proach was based on virtualization and dynamic scaling techniques to optimize the resource utilization. Virtualization is used to dynamically change which application is executed on each server. Dynamic scaling uses the optimal number of VMs that matches the workload. Our contribution in this topic is a flexible representation of an application as a graph of software components that exchange information. The approach takes the information transfer into account because the communication delay impacts on the QoS of distributed applications. Furthermore, we allow planners to limit power consumption and CO_2 emissions. The results obtained on our test cases are very important: response time can be reduced up to 6 times, 30% of power consumption can be saved, and the CO_2 emissions can be decreased up to 60 times. The algorithm developed solves instances with more than 1,000 nodes in less than a second, which is very useful for further integration in an interactive platform.

In fact, an increasing number of applications and providers require a tool to manage resources in an optimal way. Existing commercial platforms allow to deploy and monitor applications across multiple data centers, but they lack an optimization engine to help making the best decisions. Our approach would allow cloud broker users to obtain the said benefits on cost, QoS, power consumption, and CO_2 emissions.

The third stage is the placement of VMs in the servers of a data center. That is an important aspect to address because every cloud data center needs a method to define which server will host each VM. Because applications should be added and removed in an autonomic way over time, an online algorithm is needed. The scheduler proposed in this thesis places the VMs of each application by optimizing response time and power consumption. Communication among VMs is considered to place the VMs of an application in nearby servers to reduce communication delay and increase throughput among VMs. Furthermore, the proposed scheduler considers workloads that vary over the day. Applications are distributed in multiple server racks, keeping free space in each rack for the VMs that arrive in the high activity periods. An heuristic was developed to scale to more than 100,000 servers and VMs. The results of the test cases show that communication delay can be reduced by 70% compared to methods that do not consider traffic among VMs, link utilization can be reduced up to 33%, power consumption can be reduced by 4.9% when the number of VMs is changed during the day, and $1.6 millions per year can be saved in electricity in a data center of 128,000 servers.

The contributions of this thesis can be summarized as follows:

1. Definition of the Cloud Network Planning Problem (CNPP) to simultaneously decide the location of data centers and software components, routing, and assignment of link capacities.

- A flexible Mixed Integer Linear Programming (MILP) model for the CNPP that allows planners to optimize multiple objectives: cost, response time, power consumption, and CO_2 emissions.
- A tabu search heuristic for the CNPP that achieves optimality gaps between 0% and 1.95% for standard instances, and solves cases with more than 1,000 potential data center locations in less than 10 minutes.

2. Definition of an online and dynamic meta-scheduling problem to place VMs across multiple data centers and minimize cost subject to constraints on response time, power consumption, and CO_2 emissions.
   - Special consideration of communication delay between users and VMs, and among VMs themselves to reduce response time.
   - Consideration of the Power Usage Effectiveness (PUE) of data centers to reduce power consumption, and the type of energy to reduce CO_2 emissions.
3. Definition of a VM placement problem to assign VMs to servers by optimizing response time and power consumption.
   - Consideration of communication traffic among VMs to reduce communication delay and increase throughput among VMs.
   - Change the number of VMs of each application according to the workload size in each hour of the day.
   - A hierarchic tabu search heuristic that can handle hundred of thousands of servers and VMs in real time.

The work done in this thesis presents multiple avenues of future research and development. A cloud broker integrated with an optimization engine has high commercial possibilities and would have a positive impact on users, providers, and environment. The modules for meta-scheduling, deploying, and monitoring can be complemented with a workload forecasting module. That would help predicting the workload in each period of the day using techniques such as linear regression, moving average, and exponential smoothing. Given that the meta-scheduling module is dynamic, the expected workload is a parameter that would not change the model. The forecasting module should allow planners a precise calibration and validation with respect to the real workload. Going even further, the cloud broker could become a cloud market if data centers can be registered by their own, change prices, and capacities. From the point of view of infrastructure providers, pricing algorithms based on game theory would be needed to define resource prices as a function of the demand.

The planning and management of cloud computing networks present interesting challenges that will have an important impact on the resource utilization and environment. We hope that the simplicity to access and share information will have an inclusive impact for the billions of people that are still out of the digital universe and that the efforts on improving the environment will increase to improve the quality of life of current and future generations.

BIBLIOGRAPHY

[1] G. Cook, *How clean is your cloud?* Amsterdam, The Netherlands: Greenpeace International, Apr. 2012. [Online]. Available: http://greepeace.org

[2] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proc. 34th Annual International Symposium on Computer Architecture (ISCA)*, 2007, pp. 13–23.

[3] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the 2008 conference on power aware computing and systems*. USENIX Association, 2008, pp. 10–10.

[4] C. Walsworth, E. Aben, K. Claffy, and D. Andersen, "The CAIDA anonymized 2013 internet traces," 2013. [Online]. Available: http://www.caida.org/data/passive/passive_2013_dataset.xml

[5] J. C. R. Licklider, "Memorandum for members and affiliates of the intergalactic computer network," National Archives and Records Administration, Tech. Rep. RG 330-69-A-4998, Box 3, Folder 350-1, Apr 1963. [Online]. Available: http://www.dod.mil/pubs/foi/Science_and_Technology/DARPA/977.pdf

[6] L. Kleinrock, "A vision for the internet," *ST Journal of Research*, vol. 2, no. 1, pp. 4–5, 2005.

[7] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*. San Francisco, CA, USA: Morgan Kaufmann, 2004.

[8] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[9] L. Barroso and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, 2009.

[10] J. Gantz and D. Reinsel, "The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east," IDC, Tech. Rep., Dec. 2012. [Online]. Available: http://www.emc.com

[11] B. Sovacool, "Valuing the greenhouse gas emissions from nuclear power: a critical survey," *Energy Policy*, vol. 36, no. 8, pp. 2950–2963, 2008.

[12] E. Nygren, R. Sitaraman, and J. Sun, "The Akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[13] F. Glover, "Tabu search—part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[14] Í. Goiri, K. Le, J. Guitart, J. Torres, and R. Bianchini, "Intelligent placement of datacenters for internet services," in *31st IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011, pp. 131–142.

[15] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, 2003, pp. 164–177.

[16] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *IEEE Asia-Pacific Services Computing Conference (APSCC)*, 2009, pp. 103–110.

[17] K. Leal, E. Huedo, and I. M. Llorente, "A decentralized model for scheduling independent tasks in federated grids," *Future Generation Computer Systems*, vol. 25, no. 8, pp. 840–852, 2009.

[18] D. Ardagna, S. Casolari, M. Colajanni, and B. Panicucci, "Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems," *Journal of Parallel and Distributed Computing*, vol. 72, no. 6, pp. 796–808, 2012.

[19] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, 2012.

[20] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE INFOCOM*, 2010.

[21] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proc. 6th ACM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2010, pp. 15:1–15:12.

[22] W. Fang, X. Liang, S. Li, L. Chiaraviglio, and N. Xiong, "VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Computer Networks*, vol. 57, no. 1, pp. 179–196, 2013.

[23] F. Larumbe and B. Sansò, "A tabu search heuristic for the location of data centers and software components in green cloud computing networks," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 22–35, 2013.

[24] ——, "Green cloud broker: dynamic virtual machine placement across multiple cloud providers," *IEEE Transactions on Cloud Computing*, 2013, submitted for publication.

[25] J. Park, "Open Compute project: data center mechanical and electrical specification," Apr. 2011. [Online]. Available: http://opencompute.org

[26] D. A. Menasce and V. A. F. Almeida, *Capacity planning for web services: metrics, models, and methods*. Upper Saddle River, NJ, USA: Prentice Hall, 2002.

[27] H. A. Linstone and M. Turoff, *The delphi method*. MA, USA: Addison-Wesley Reading, 1975.

[28] G. Rowe and G. Wright, "The Delphi technique: past, present, and future prospects—introduction to the special issue," *Technological Forecasting and Social Change*, vol. 78, no. 9, pp. 1487–1490, 2011.

[29] S. Chang, S. Patel, and J. Withers, "An optimization model to determine data center locations for the army enterprise," in *World's Premier Military Communications Conference (MILCOM)*, Oct. 2007, pp. 1–8.

[30] X. Dong, T. El-Gorashi, and J. Elmirghani, "Green IP over WDM networks with data centers," *Journal of Lightwave Technology*, vol. 29, no. 12, pp. 1861–1880, 2011.

[31] F. Larumbe and B. Sansò, "Cloptimus: a multi-objective cloud data center and software component location framework," in *1st IEEE International Conference on Cloud Networking (CLOUDNET)*, Nov. 2012, pp. 23–28.

[32] M. Covas, C. Silva, and L. Dias, "Multicriteria decision analysis for sustainable data centers location," *International Transactions in Operational Research*, vol. 20, no. 3, pp. 269–299, May 2013.

[33] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[34] W. Yu, "ELECTRE TRI: aspects méthodologiques et manuel d'utilisation," Document - Université de Paris-Dauphine, LAMSADE, Tech. Rep. 92-74, 1992.

[35] Facebook, "Luleå data center," Apr. 2013. [Online]. Available: http://facebook.com/luleaDataCenter

[36] Z. Drezner and H. Hamacher, *Facility location: applications and theory.* Germany: Springer, 2004.

[37] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," in *Workshop on Compilers and Operating systems for Low Power (COLP)*, 2001, pp. 182–195.

[38] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 3, no. 1, p. 1, 2008.

[39] Amazon, "Amazon Elastic Compute Cloud (EC2)," Jan. 2013. [Online]. Available: http://aws.amazon.com/ec2/

[40] OpenStack, "Heat: a template based orchestration engine for OpenStack," 2013. [Online]. Available: http://www.openstack.org

[41] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *4th IEEE International Conference on Cloud Computing (CLOUD)*, 2011, pp. 500–507.

[42] OpenStack, "Open source software for building private and public clouds," 2013. [Online]. Available: http://www.openstack.org

[43] A. Corradi, M. Fanelli, and L. Foschini, "VM consolidation: a real case based on OpenStack cloud," *Future Generation Computer Systems*, to be published.

[44] T. White, *Hadoop: the definitive guide.* Sebastopol, CA, USA: O'Reilly Media, 2012.

[45] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: state of the art and open problems," School of Computing, Queen's University, Tech. Rep. 2006-504, 2006.

[46] D. Franceschelli, D. Ardagna, M. Ciavotta, and E. Di Nitto, "SPACE4CLOUD: a tool for system performance and costevaluation of cloud systems," in *Proc. 1st ACM International workshop on multi-cloud applications and federated clouds*, 2013, pp. 27–34.

[47] B. Kantarci, L. Foschini, A. Corradi, and H. T. Mouftah, "Inter-and-intra data center VM-placement for energy-efficient large-scale cloud systems," in *IEEE Globecom Workshops (GC Wkshps)*, 2012, pp. 708–713.

[48] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, no. 1, pp. 3–22, 2009.

[49] Standard Performance Evaluation Corporation, "Specweb2005," 2013. [Online]. Available: http://www.spec.org/web2005/

[50] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani, "Scaling memcache at Facebook," in *Proc. 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr. 2013, pp. 385–398.

[51] VMware, "VMware vSphere 4: The CPU scheduler in VMware ESX 4," VMWare, Tech. Rep., 2010. [Online]. Available: http://www.vmware.com/resources/techresources/10059

[52] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE Transactions on Services Computing*, vol. 3, no. 4, pp. 266–278, 2010.

[53] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[54] A. Verma, G. Dasgupta, T. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proc. USENIX Annual technical conference*, 2009, pp. 28:1–28:14.

[55] B. Addis, D. Ardagna, B. Panicucci, and L. Zhang, "Autonomic management of cloud service centers with availability guarantees," in *3rd IEEE International Conference on Cloud Computing*, 2010, pp. 220–227.

[56] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang, "Energy-aware autonomic resource allocation in multitier virtualized environments," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 2–19, 2012.

[57] B. Addis, D. Ardagna, B. Panicucci, M. Squillante, and L. Zhang, "A hierarchical approach for the resource management of very large cloud platforms," *IEEE Transactions on Dependable and Secure Computing*, to be published.

[58] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005, pp. 273–286.

[59] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges," in *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2010.

[60] M. Armbrust, A. Fox, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: a Berkeley view on cloud computing," University of California at Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.

[61] R. Brown, E. Masanet, B. Nordman, B. Tschudi, A. Shehabi, J. Stanley, J. Koomey, D. Sartor, P. Chan, J. Loper, S. Capana, B. Hedman, R. Duff, E. Haines, D. Sass, and A. Fanara, "Report to Congress on server and data center energy efficiency," *Public law*, vol. 109, p. 431, 2007.

[62] R. Buyya, R. Ranjan, and R. Calheiros, "InterCloud: utility-oriented federation of cloud computing environments for scaling of application services," in *Proc. 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*. Springer, May 2010, pp. 328–336.

[63] F. Larumbe and B. Sansò, "Location and dynamic provisioning problems in cloud computing networks," in *Communication infrastructures for cloud computing: design and applications*, M. Hussein and B. Kantarci, Eds. IGI Global, Sep. 2013.

[64] H. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 1, pp. 85–93, 1977.

[65] L. Rao, X. Liu, L. Xie, and W. Liu, "Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[66] N. Chowdhury, M. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proc. IEEE INFOCOM*, 2009, pp. 783–791.

[67] A. Geoffrion and R. Bride, "Lagrangean relaxation applied to capacitated facility location problems," *AIIE Transactions*, vol. 10, no. 1, pp. 40–47, 1978.

[68] A. Klose and S. Gortz, "A branch-and-price algorithm for the capacitated facility location problem," *European Journal of Operational Research*, vol. 179, no. 3, pp. 1109–1125, 2007.

[69] H. Pirkul, "Efficient algorithms for the capacitated concentrator location problem," *Computers & operations research*, vol. 14, no. 3, pp. 197–208, 1987.

[70] M. Plante and B. Sansò, "A typology for multi-technology, multi-service broadband network synthesis," *Telecommunication Systems*, vol. 19, no. 1, pp. 39–73, 2002.

[71] F. Larumbe and B. Sansò, "Optimal location of data centers and software components in cloud computing network design," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Workshop on Cloud Computing Optimization (CCOPT)*, May 2012, pp. 841–844.

[72] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya *et al.*, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in Computers*, vol. 82, no. 2, pp. 47–111, 2011.

[73] F. Glover, "Tabu search—part II," *ORSA Journal on Computing*, vol. 2, no. 1, pp. 4–32, 1990.

[74] A. Hertz and D. Werra, "Using tabu search techniques for graph coloring," *Computing*, vol. 39, no. 4, pp. 345–351, 1987.

[75] T. Brinkhoff, "City population," Jan. 2013. [Online]. Available: http://www.citypopulation.de/

[76] Internet World Stats, "Usage and population statistics," Jan. 2013. [Online]. Available: http://www.internetworldstats.com

[77] Google, "Google Map," Jan. 2013. [Online]. Available: http://map.google.com

[78] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet inter-domain traffic," in *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, 2010, pp. 75–86.

[79] Telegeography, "Executive summary," Jan. 2013. [Online]. Available: http://www.telegeography.com

[80] Dell, "Online store," Jan. 2013. [Online]. Available: http://www.dell.com

[81] U.S. Energy Information Administration, "Independent statistics & analysis," 2011. [Online]. Available: http://www.eia.gov/todayinenergy/detail.cfm?id=4530

[82] "SMART 2020: Enabling the low carbon economy in the information age," The Climate Group, Jun. 2008. [Online]. Available: www.smart2020.org/_assets/files/02_Smart2020Report.pdf

[83] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, and C. Sheridan, "MODAClouds: a model-driven approach for the design and execution of applications on multiple clouds," in *ICSE Workshop on Modeling in Software Engineering (MISE)*. IEEE, 2012, pp. 50–56.

[84] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan, "OPTIMIS: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, 2012.

[85] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán, "The Reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4:1–4:11, 2009.

[86] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[87] A. Dhamdhere and C. Dovrolis, "The internet is flat: modeling the transition from a transit hierarchy to a peering mesh," in *Proc. 6th ACM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2010, pp. 21:1–21:12.

[88] B. Fitzpatrick, "Distributed caching with memcached," *Linux journal*, no. 124, pp. 72–74, 2004.

[89] D. Meisner, B. Gold, and T. Weinisch, "Powernap: Eliminating server idle power," in *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, 2009, pp. 205–216.

[90] A. Gandhi, M. Harchol-Balter, and M. A. Kozuch, "The case for sleep states in servers," in *Proc. 4th ACM Workshop on Power-Aware Computing and Systems (HotPower)*, 2011, pp. 2:1–2:5.

[91] T. Sato, "ext4 online defragmentation," in *Proc. Linux Symposium*, vol. 2, 2007, pp. 179–186.

[92] U.S. Energy Information Administration, "Average retail price of electricity to ultimate customers by end-use sector," May 2013. [Online]. Available: http://www.eia.gov/todayinenergy/detail.cfm?id=4530

[93] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, 2008, pp. 63–74.

[94] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, 2009, pp. 51–62.

[95] MercoPress, "Brazil: internet providers will have to store domestic communications in the country," Jul. 2013. [Online]. Available: http://en.mercopress.com

[96] OPower, "Web site," Jul. 2013. [Online]. Available: http://opower.com/

[97] Arbor Environmental Alliance, "Web site," 2013. [Online]. Available: http://www.arborenvironmentalalliance.com/